Final Report for AOARD Grant 104138

"Contextual Awareness for Robust Robot Autonomy"

December 30, 2013

Name of Principal Investigators (PI and Co-PIs): Reid Simmons

- e-mail address : reids@cs.cmu.edu
- Institution : Robotics Institute, Carnegie Mellon University
- Mailing Address : 5000 Forbes Avenue, Pittsburgh PA 15213
- Phone : 412-268-2621
- Fax : 412-268-7350

Period of Performance: 08/24/2010 – 08/23/2013

Abstract:

Contextual awareness refers to having autonomous robots reason about their capabilities and limitation and improve on those limitations through the help of others. This project explored three areas of contextual awareness – detecting when anomalous behaviors occur, reacting to situations where plans are failing, and learning new plans through human demonstration. Each of these areas is important in achieving robust, reliable robot autonomy. The work on detecting anomalous behavior focused on finding subtle anomalies that could not be detected from single events; the work on reacting to failing plans focused on deciding when to switch between risk-neutral and risk-seeking policies, for domains in which the goal is to achieve above a certain threshold of reward; and the work on learning new plans focused on complex manipulator trajectories, where multiple human examples are combined so as to smooth out noise in the examples without losing important details. The first and third areas were demonstrated using actual robots; the second area was demonstrated using a video game simulator.

Introduction:

Contextual awareness refers to having autonomous robot systems reason about their capabilities and limitation and improve on those limitations through the assistance of others. The idea is for robots to become more robust through a better understanding of how they interact with the environment, and how people interact with the environment. We have investigated three areas of contextual awareness that we believe are critical in achieving robust behavior in autonomous robots.

1. **Awareness of the robot's behaviors** – specifically, investigating how to detect anomalous behaviors that are too subtle to detect from a single perceptual event. The idea is that certain problems, such as wheel slippage, are hard to distinguish from noisy sensor readings, but show up when aggregated over time. By determining the contexts in which these anomalies occur, the robot can reason more accurately about its behavior, and avoid anomalous regions of the state space, if necessary.
2. **Awareness of the ability to achieve the robot's goals** – specifically, investigating when to decide that one plan is failing and to switch to another, more appropriate, strategy. The idea is that the plan that is optimal, in general, may not be best under certain circumstances. This idea was investigated in domains where the agent must achieve a given reward threshold – anything reward below that threshold is equivalent to failure. This includes competitive games, such as sports, where losing is everything, and it really does not matter by how much one loses. By being aware of the contexts in which one plan may be more desirable than another, the robot can intelligently decide how best to achieve its goals in certain, perhaps unusual, situations.
3. **Awareness of the need for assistance** – specifically, we investigated the ability of a robot to utilize human examples of achieving a task in order to learn how to reliably achieve the task itself. Here, the human provides example trajectories of the robot performing a complex manipulation task. The robot uses the example to plan new trajectories to achieve the goal, smoothing out noise and disturbances present in the human example, while maintaining important details that

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **16 JAN 2014**	2. REPORT TYPE **Final**	3. DATES COVERED **24-08-2010 to 23-08-2013**

4. TITLE AND SUBTITLE **Contextual Awareness for Robust Robot Autonomy**	5a. CONTRACT NUMBER **FA23861014138**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Reid Simmons**	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,5000 Forbes Avenue,Pittsburgh, PA 15213-3890,United States,PA,15213-3890**	8. PERFORMING ORGANIZATION REPORT NUMBER **N/A**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **AOARD, UNIT 45002, APO, AP, 96338-5002**	10. SPONSOR/MONITOR'S ACRONYM(S) **AOARD**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) **AOARD-104138**

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Contextual awareness refers to having autonomous robots reason about their capabilities and limitation and improve on those limitations through the help of others. This project explored three areas of contextual awareness ? detecting when anomalous behaviors occur, reacting to situations where plans are failing, and learning new plans through human demonstration. Each of these areas is important in achieving robust, reliable robot autonomy. The work on detecting anomalous behavior focused on finding subtle anomalies that could not be detected from single events; the work on reacting to failing plans focused on deciding when to switch between risk-neutral and risk-seeking policies, for domains in which the goal is to achieve above a certain threshold of reward; and the work on learning new plans focused on complex manipulator trajectories, where multiple human examples are combined so as to smooth out noise in the examples without losing important details. The first and third areas were demonstrated using actual robots; the second area was demonstrated using a video game simulator.

15. SUBJECT TERMS

Artificial Intelligence, Robotics, computational intelligence, Autonomous Agents and Multi-Agent Systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**40**	

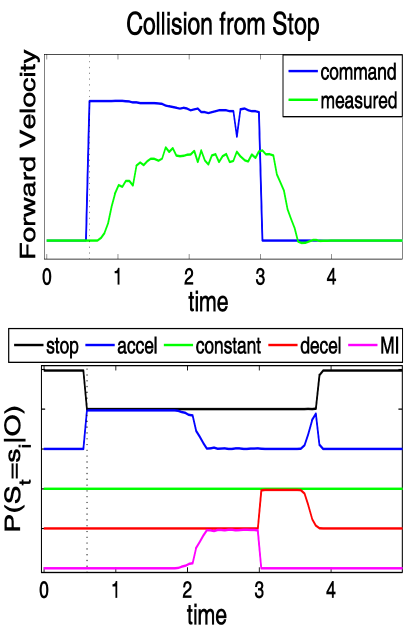
are common in all/most of the examples. By taking advantage of the facility that people have in performing tasks, even though they do not perform them identically each time, the robot can learn a robust policy for performing such tasks.

Overall, our three investigations produced techniques and algorithms that can improve the robustness of autonomous systems. The techniques have been tested and demonstrated using actual robots and realistic simulators. While they have not been integrated, they are naturally complementary, and we expect that their integration would produce synergistic effects.

Approach:

The focus on awareness of the **robot's behaviors** was to detect subtle and unexpected failures. More specifically, we wanted to detect anomalies – differences between the expected behavior of the robot and its actual behavior. The idea was that it would be difficult, or impossible, to enumerate all possible fault behaviors, but that modeling the nominal behavior would be rather straightforward. Then, if the robot would estimate that its current behavior differed significantly from its commanded behavior, that would indicate a problem. While we might not understand what actually caused the problem (but see below), it is still a good first step to acknowledge that a problem exists. At the very least, the robot could stop (or go into a safe mode) and communicate back to a human for help. This would be much more desirable than continuing to operate in a potentially unsafe manner.

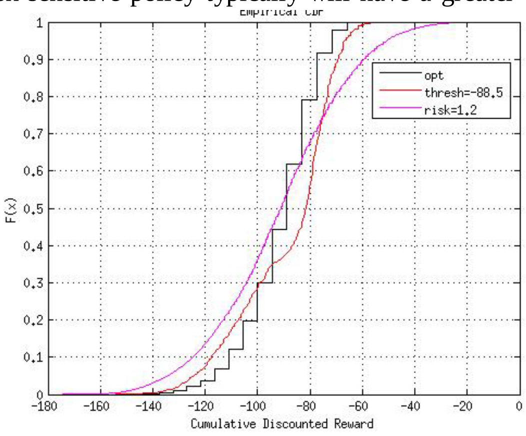
Our first investigation into this area was to learn to detect *motion interference*, where something is interfering with the normal navigation of a mobile robot [Mendoza 2012a]. The idea is to learn a Hidden Markov Model (HMM) that described the nominal behavior of an autonomous mobile robot (figure at left). The HMM was trained using hand-labeled data, with the inputs being motion commands and wheel encoder data. The system learned the transition probabilities from each state (e.g., stop) to the other states (e.g., accelerate) plus the observation probabilities of all states. In addition, a special "motion interference" (MI) state was defined as an anomalous state, with the transition probabilities to that state being a tunable parameter. During execution, the system would estimate the probabilities for each state, and would stop the robot if the MI state ever became the most probable. By changing the probability of the transition to the MI state (which is not actually readily measurable, since it occurs so infrequently, the precision and recall of the system could be varied. The best value tested gave a precision of 1 and a recall of 0.93, with a median time of 0.36 seconds to detect an anomaly (where sensor input was at 20Hz). The figure on right shows an example where the robot hit an obstacle (see figure above left) at time 2. The system predicts the robot is stopped until about 0.5, then accelerates until time 2, then is in collision until time 3, then decelerates until time 3.7, then stops. There is a slight blip at time 3.8 where the system is thought to be accelerating – this is probably because it moves forward slightly as it disengages from the obstacle.



Our next investigation was to determine regions of the robot's state space where anomalies typically occur. The idea is to detect anomalies that are too subtle to be found using only a single event, or in a single state, but may be detected in the aggregate, both over time and in a region of the state space [Mendoza 2012b]. For instance, the robot might be found to slip more in certain hallways, or only when turning left, or only when going between 75-90 cm/sec. Again, we use a model of nominal behavior, and define a residual function that is (close to) zero during nominal behavior. If the system them determines that the sample mean of the residual function is greater than some epsilon, it indicates an anomalous situation (again, we do not know what caused it, only that it is not nominal). Much of the work in this area has been in researching how to efficiently aggregate parts of the state space to find anomalous regions. The work reported in [Mendoza 2012b] used axis-aligned hyper-rectangles, which were indeed efficient to calculate, but of limited representational expressiveness. More recently, we have looked into representing the regions using hyper-ellipsoids and have developed a method that combines RANSAC and gradient ascent for picking the regions that are most likely to include the anomaly [Mendoza 2014]. We are currently working to validate these ideas by porting this approach to a mobile robot that autonomously navigates the corridors of the CMU Computer Science building.

The focus on awareness of the **ability to achieve the robot's goals** was to estimate when an agent is likely not making sufficient progress towards it goals and to try to recover from such situations. This is important because, often, robots will not detect such problems until it is too late to recover, if at all. We investigated a particular class of domains, one in which the agent incurs costs for taking actions and receives rewards for achieving certain goals, and the objective is to exceed a given reward threshold. For instance, one might have the goal of making a certain number of deliveries within 30 minutes, or having the point differential in a sports game be positive in the agent's favor. Often, in such domains, humans will take risks (e.g., a "hail Mary pass" in football, trying risky military tactics if the situation appears dire) if they believe that they would not otherwise achieve their objectives. The idea is that it does not matter how much less than the threshold the agent achieves, as long as it still has a chance of not falling below.

Our basic approach is to model the task as a Markov Decision Process (MDP) and to track the anticipated reward that the robot will receive (current cumulative reward plus expected future reward) and compare that with the threshold value [Kane, 2012]. When the threshold is crossed, the robot switches policies (e.g., to a more, or less, risky policy). Specifically, given the MDP, we compute both a risk-neutral and risk-sensitive policy. The risk-sensitive policy typically will have a greater downside (more chance of receiving a lower value), but also a greater upside (greater probability of getting a higher value). We run many simulations of each policy to construct a CDF curve showing the cumulative probability of achieving at least some given value (figure on right, where the lower the curve the better it is). Representing the complete reward distribution differs from other work that characterizes the reward either in expectation (mean) or mean and variance. Having the complete distribution enables us to reason about the tails of the distribution, which is critical in determining which policy is better in a given situation.

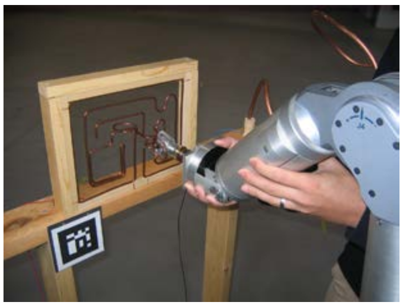


The objective, then, is to maximize the probability that the robot will achieve at least a certain minimum threshold value. At each step of execution, our algorithm determines the probability that each policy will lead to a value above the threshold, given the current state and the reward accumulated, so far. The policy with the highest probability is chosen for each step. In practice, this does exactly what we would like to see – when things are going well, the risk-neutral policy is chosen; when the predicted future reward gets closer to the threshold value, the risk-sensitive policy is seen as more attractive and is chosen. If using the risk-sensitive policy produces a "good" outcome, the system switches back to the risk-neutral policy. The results are illustrated in the figure above – the switching approach (labeled thresh=-88.5, which is the given threshold for this particular experiment) is much more likely than the risk-neutral or risk-sensitive policy, alone, to exceed the given threshold. Even though, when it fails, it achieves worse reward than the risk-neutral policy, this is not really a problem, since the formulation of this domain stipulates that there is no penalty for how far below the threshold one falls. Similarly, it is worse that the risk-sensitive policy, starting at around reward -78, but again that is not an issue, since it also does not matter how far above the threshold one gets, as long as one is above.

We have tested this approach extensively with a simple "pizza delivery" domain (must make the delivery in a given amount of time), where there are expensive, risky driving actions and more

conservative, cheaper actions. Depending on the threshold chosen, the switching strategy fails between 10-20% less than using the risk-neutral policy alone (and does even better than using the risk-sensitive policy alone). We also tested the approach in a much more complex, realistic domain, that of the Infinite Mario video game simulator (figure on right). Here, we collected reward statistics both using an MDP learned from having a human play the game and from having the agent play the game. Even though the models were not accurate (e.g., did not accurately represent the number of coins or monsters that were available), our approach still failed between 5-15% less than using the risk-neutral strategy found be solving the learned MDP. Finally, we compared our approach against a more straightforward approach of explicitly representing the cumulative reward, so far, in the state space, and solving the augmented MDP. This augmented MDP has significantly more states than the original MDP, and takes exponentially longer to solve. In addition, it must be solved for each threshold value, whereas in our approach, the MDPs need to be solved only once, for each risk value. In tests, the augmented MDP failed only 7% less than our switching strategy (in the pizza delivery domain), which we consider a good tradeoff, given the amount of computation needed to solve the augmented MDP.

In recent months, we have shifted focus to looking at recovering from unexpected failures. This is somewhat of a paradox – how can you figure out how to recover from something that is truly unexpected (i.e., unmodeled). Our approach to this conundrum is to have many models, at multiple levels of detail, and choose which model to plan in dynamically. For instance, one would start doing path planning in a geometric state space (maybe just taking X,Y into account) and then move to a more detailed geometric model (X,Y,Z or X,Y,theta), depending on the type of failure encountered. If the geometric model was still not detailed enough, one would move to dynamical models, adding velocities and/or accelerations, as appropriate. This approach provides the ability to plan using the model that is most appropriate to the situation, but still enables the agent to "dig deeper" when the situation warrants. We are in the process of implementing this approach in the domain of autonomous robot navigation using a realistic, dynamical simulator, and are currently preparing a paper on the subject to submit to a robotics conference.



The focus on awareness of the **need for assistance** was to use human examples to learn how to plan precise, complex trajectories for a robotic manipulator. The idea is to avoid having to model all the geometric and non-geometric (task) constraints by having people show the robot how to perform a task. The robot generalizes the examples to remove the extraneous noise and jitter that often accompanies human kinesthetic demonstrations (figure on right). The motivation for this area of research was an earlier project where we found that programming the various constraints that the manipulator must obey was arduous, yet people could easily teleoperate the robot into place. The project developed techniques, based on dimensionality reduction [Melchior 2010], to find the common parts of different example trajectories and then used those correspondences to plan novel trajectories that maintained the important commonalities of the human examples [Melchior 2012]. By using multiple examples, the algorithm was able to smooth out imperfections in the examples while maintaining relevant task features. The approach was context-aware, in that it was correctly able to deal with examples that got near to, or even crossed, one another (either in state or configuration space), which previous techniques were not able to handle very robustly. We also developed several quantitative metrics to evaluate the quality of plans produced, including the overall smoothness and path length. In general, the generated plans were found to be significantly smoother and somewhat shorter than the examples provided by the human teachers, showing that our approach is able to use noisy examples to produce high-quality trajectories.

Sometimes, the human demonstrations followed qualitatively different strategies (for instance, sometimes going around an obstacle to the right, and sometimes to the left). We used "active learning" to help the robot better handle such situations. The idea is for the robot to reason about those parts of the state space where the example trajectories may be ambiguous or inconsistent, and ask the human teacher for help in disambiguating. Specifically, we developed an algorithm, based on min-cuts, to determine where "bifurcations" occur (qualitative splits in the strategies for achieving a task), demonstrating to the user plans that illustrate both branches of the bifurcation, and asking the user for his/her preference (or indifference to) the two trajectories. Based on the human's answer, the robot will either prefer one branch over the other when generating plans in the future, or may ask the user for additional examples to remove any existing ambiguities. The algorithm was tested on 18 participants with varying degrees of expertise with robots. Results indicate that participants found the learning by demonstration methodology very easy to use, producing effective, high-quality plans, especially for novice users.

List of Publications and Significant Collaborations:

a) papers published in peer-reviewed journals - *none*
b) papers published in peer-reviewed conference proceedings
"Graph-based Trajectory Planning through Programming by Demonstration," N.A. Melchior and R. Simmons, In *Proceedings of International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, October 2012
"Motion Interference Detection in Mobile Robots," J.P. Mendoza, M. Veloso and R. Simmons, In *Proceedings of International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, October 2012
"Mobile Robot Fault Detection based on Redundant Information Statistics," J.P. Mendoza, M. Veloso and R. Simmons, In *IROS Workshop on Safety in Human-Robot Coexistence and Interaction*, Vilamoura, Portugal, October 2012
"Risk-Variant Policy Switching to Exceed Reward Thresholds," B. Kane and R. Simmons, ICAPS, Sao Paulo Brazil, June 2012
"Dimensionality Reduction for Trajectory Learning from Demonstration," N. A. Melchior and R. Simmons. In Proceedings of IEEE International Conference on Robotics and Automation, May 2010
c) papers published in non-peer-reviewed journals and conference proceedings
"Graph-based Trajectory Planning through Programming by Demonstration," Nik Melchior, PhD Thesis, CMU RI-TR-11-40, August 2012 (technical report)
d) conference presentations without papers – *none*
e) manuscripts submitted but not yet published
"Early Detection of Anomalous Clusters for Task Execution Monitoring," J.P. Mendoza, M. Veloso and R. Simmons, submitted to International Conference on Robotics and Automation, 2014
f) interactions with industry or with Air Force Research Laboratory scientists - *none*

Graph-based Trajectory Planning through Programming by Demonstration

Nik A. Melchior and Reid Simmons[1]

***Abstract*— As robots are utilized in a growing number of applications, the ability to teach them to perform tasks safely and accurately becomes ever more critical. *Programming by demonstration* offers an expressive means for teaching while being accessible to domain experts who may be novices in robotics. This work investigates a programming by demonstration approach to learning motion trajectories for robotic manipulator tasks. Using a graph constructed to determine correspondences between multiple imperfect demonstrations, the robot learner plans novel trajectories that safely and smoothly generalize the teacher's behavior, while attenuating those imperfections. The learner also actively detects instances of diverging strategy between examples, requesting advice for resolving these ambiguities. We demonstrate our approach in example domains with a 7 degree-of-freedom manipulator.**

I. Introduction

Robots are becoming more common in many domains: They are used as tools for manufacturing, instruments for surgery, and toys for consumers. As their areas of application expand, it becomes increasingly critical to reliably transfer task knowledge to the robot. While domain experts often can be found who understand the task, they may know nothing about programming robots. *Programming by demonstration* (PbD) is an approach that facilitates knowledge transfer from a domain expert to an autonomous system. It provides an intuitive approach for someone skilled in performing a task to teach a robot to perform that task without having to learn to program the robot. Conversely, it does not require a robotics expert to become skilled in the task.

A primary difficulty in PbD approaches to robotic manipulation is representing and understanding the constraints imposed by the physical world. *Geometric* constraints, including the locations of physical obstacles, are the most obvious issues, since detecting the locations of objects can be difficult for current sensor technologies. Vision or LIDAR sensors must be located to observe the entire environment, without suffering from occlusion due to the objects or the robot itself. Alternately, the robot may be provided with an *a priori* model of the objects of interest, but constructing this model may require the skills of a robotics specialist.

Even if a model could be constructed to provide the robot with knowledge of obstacles, *non-geometric* task constraints must also be considered. For example, a robot carrying a cup of liquid must maintain its end-effector orientation to avoid spilling. Similarly, a robot routing cable around complex objects may need to follow a particular path to avoid snagging the cable. While a detailed physical simulation may be able to detect violations of such constraints, it would be challenging for a novice robot user to communicate them to a planner. However, if a user is able to demonstrate successful strategies for completing the task, he does not need to articulate the criteria that he is optimizing in a manner comprehensible to the robot.

Thus, PbD enables domain experts to teach robots about geometric and non-geometric task constraints without explicitly formalizing those constraints. The robot then create novel plans respecting the constraints, generalizing the actions performed by the teacher. In this work, we present a method for planning novel motion trajectories for robot manipulators based on demonstrations. Our approach learns to perform fixed, repetitive tasks in the presence of static obstacles, with minor variations due to uncertainty in both initial conditions and the ability to follow a trajectory precisely. In particular, we do not assume that example demonstrations are flawless, but rather that they contain jitter, non-optimal movements, and inconsistencies between examples. The robot must therefore discern the essential motions common to multiple examples and incorporate them in novel plans. In addition, we assume that the examples may differ in essential characteristics, making it difficult for the robot to combine the trajectories into a single, learned strategy. In such cases, the robot must refine its knowledge by actively requesting additional information from the teacher, in order to differentiate the manipulation strategies.

Our approach relies on a *neighbor graph* over the demonstrations, a representation that relates discretely sampled points from example trajectories. Our previous work [1], summarized in Section III, details a method to build such a data structure. The planning algorithm contributed in the present work uses this neighbor graph to create new plans that are guaranteed to be safe (avoiding obstacles) as long as the demonstration trajectories are also safe. Each point in a new plan is constructed using a set of neighbors from the demonstration set, refined to reduce jitter and minor inconsistencies, while maintaining the essential (common) characteristics of the demonstration trajectories. The planning algorithm also detects demonstrations that cluster into qualitatively distinct trajectories (for instance, circumventing an obstacle by two entirely different routes). In such cases, the robot requests additional advice from the operator to determine which strategy should be preferred. Experimental results in Section V demonstrate our approach in two different domains: The robot learner is able to safely trace a smooth path through a maze (with a single solution) and across a planar surface with multiple obstacles and multiple

[1]Robotics Institute of Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, USA; {nmelchio, reids}@cs.cmu.edu

demonstrated routes between them. We also present results from a user study that indicates how well novices can use our approach to train the robot.

II. Related Work

Approaches to programming by demonstration differ widely in their use of prior knowledge and the models used to create new plans. Simplified motion or world models [2], [3], [4] and even traditional motion planning [5] can be used to plan between demonstrated states if an environment model is available. Use of the model may alternately be limited to collision testing plans created by other methods [6], [7].

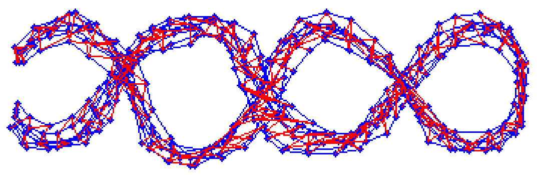
Many approaches collect similar actions into libraries, which can be applied to create motion plans for new tasks or situations. [8], [9]. Others attempt to learn dynamical models of the system using GMMs [10], Neural Networks [11], or other statistical feedback techniques [12]. These models may then be used to create new plans. Feature-based policy-learning approaches such as that of Argall [13], Chernova [14], and Ratliff [15] also create results that are applicable to new tasks and domains.

Our approach is most similar in spirit to data-driven approaches such as the interpolation-based methods of Ude [16], Lee [17] and Aleotti [6] or flow-field techniques developed by Mayer [18] and Drumwright [19]. These approaches, like ours, focus on learning entire tasks (or portions of tasks) at a time, rather than computing an action based solely on local features. Among similar approaches, only Delson's work [20], though, explicitly considers safety during the generation of novel trajectories. This approach is most completely developed in two dimensions, but an extension to three dimensions is presented in [21].

Dimensionality reduction approaches attempt to discover a lower dimensional representation of the demonstrated motions that retains the crucial features of the examples. While some such approaches use techniques, such as PCA, that retain as much variance as possible [22], others attempt to directly estimate correspondences between portions of example trajectories [23]. Rather than optimize for variance across the entire dataset, this alternative, which builds upon Isomap [24], optimizes for distances between points in the dataset. This approach requires a neighbor graph connecting points that are considered qualitatively similar. Distances between neighboring points are calculated directly using the metric of choice (e.g. Euclidean), but all other pairwise distances are calculated as the sum of shortest-path link distances through the neighbor graph. These *geodesic* distances indicate the nearness of points in terms of task execution, and provide a basis for determining which points can reasonably be clustered and interpolated.

III. Background

Our previously reported work [1] developed a neighbor-finding technique that used a set of heuristics to produce a neighbor graph suitable for reduced-dimensionality planning. Planning in a low-dimensional *latent* space is attractive because the arrangement of trajectory points in this space


(a) Demonstration *slalom* trajectories (blue) with neighbor links (red).

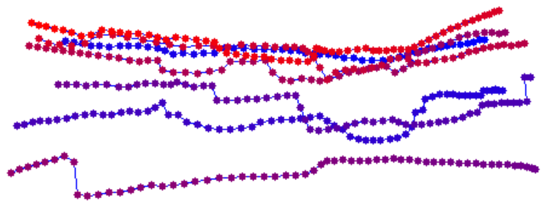

(b) Slalom trajectories in the latent space.

Fig. 1. Slalom trajectories.

is expected to correlate with the semantics of the task. For example, the trajectories of Figure 1(a) cross over themselves in work space, but are "unrolled" in the latent space (Figure 1(b)). The start and goal points, previously close to one another, are now at opposite ends of the horizontal axis. Thus, this dimension represents progress through the task, while the vertical dimension separates distinct demonstrations. Planning through this space would seem to be intuitive since movement in each direction has a semantic explanation.

Unfortunately, we have found that this and similar approaches to dimensionality reduction do not lead to robust planning. For one, distances in the neighbor graph are particularly susceptible to spurious neighbor links. While these algorithms are robust to a large number of false negatives (missing neighbor links), even a single false positive link can have disastrous effects on the embedding [25]. This is because spurious neighbor links create "short circuit" connections between unrelated portions of trajectories, thus lowering the geodesic distances between points in semantically distant regions. This forces the embedding to retain the proximity of these regions in the low-dimensional latent space, introducing false semantic relevance between points.

While our neighbor-finding approach mostly avoids the problem of spurious neighbor links [1], another fundamental problem is that *lifting* novel trajectories from the latent space back to the work (Euclidean or configuration) space often produces second-order discontinuities. Lifting is typically performed by projecting discrete points sampled from the trajectory in the latent space into the work space. Individual query points are lifted by considering nearby sample points whose corresponding high-dimensional points are known. The high-dimensional point corresponding to the query is calculated by interpolating between the high-dimensional neighbor points, weighted by their distance from the query in the low-dimensional space. Unfortunately, non-linear em-

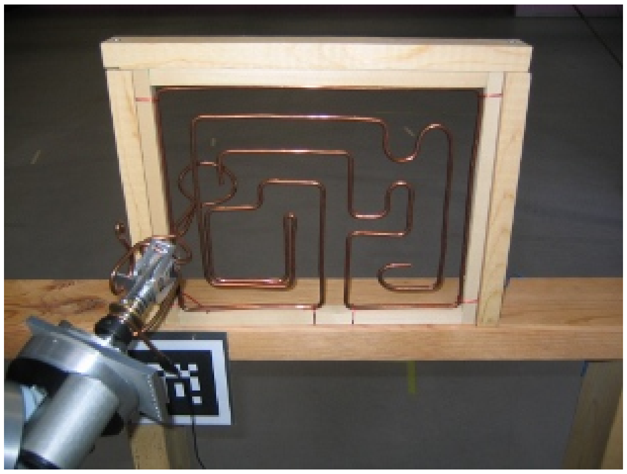


Fig. 2. The WAM manipulator traversing a wire maze. The fiducial in the lower-left is used to detect the location of the rig relative to the robot.

beddings, such as Isomap [24], do not guarantee smoothness of lifted trajectories, even when points are densely sampled from a smooth latent-space trajectory. Moreover, because dimensionality reduction is not *injective* (multiple regions of the work space can map to the same regions of the latent space), there may be ambiguities in lifting points back to the work space. These problems are actually exacerbated as additional example trajectories are provided to the learner. More examples produce a more cluttered latent space with more interconnections between neighbor points. This creates additional opportunities for discontinuities in lifted trajectories. Clearly, decreasing performance with an increasing amount of information is not a desired attribute of a planner.

These problems are clearly illustrated in our wire maze domain (Figure 2). Here, participants were asked to guide a 7-DOF Barrett WAM arm through a wire maze, with the arm in passive gravity-compensation mode. Figure 3 displays workspace traces of six demonstration trajectories (in purple) beginning at the green points near the bottom of the image and ending at the red points. The latent space embedding created by our algorithm is shown on the right. As expected, this embedding essentially "unrolls" the trajectories, stretching them out so that task time, or progression through the maze, maps from left to right on the horizontal axis. The vertical axis provides a dimension for variation between examples. The blue line represents a candidate plan, created as a series of line segments stretching from the start to the goal in the latent space. Discontinuities result, however, when this plan is lifted back to the original workspace in the left image. Although a post-processing step could be applied to smooth the planned trajectory, it is not clear whether a smoothed version would follow the nuances of the demonstration trajectories, or even maintain safety.

IV. Approach

The approach described in this paper addresses these issues by planning directly in the work space. We still use the neighbor graph to determine correspondences between trajectory points, but rather than planning in the latent space, interpolation and distance calculations are performed in the original (higher-dimensional) work space. Our planner interpolates between the multiple demonstration trajectories, eliding inconsistencies (errors and suboptimal motions) while maintaining the essential geometric characteristics of the demonstrations.

As the first step in neighbor finding, we uniformly sample each trajectory at fixed distances in the work space. The sampling is done to compensate for temporal differences in the demonstrations (this assumes that dynamics are not fundamental to task achievement). Then, heuristics (described in detail in [1]) are used to find correspondences between neighboring points. The heuristics attempt to find local coherence between trajectories; for instance, encoding the idea that corresponding points are more likely to have nearby predecessors that also correspond. The result is a graph where connections between trajectories (usually) indicate semantically meaningful matches, while points that remain unmatched typically result from imperfections in the demonstrations, such as jitter or small detours.

The planning algorithm proceeds iteratively by calculating an action to take based on a set of neighbor points and then choosing a new set of neighbor points based on that action, until a goal state is reached. The initial set of neighbor points is chosen simply as those nearest the start state of the robot. The action is computed as a locally weighted average of the actions associated with each point in the neighbor set (Fig. 4(b)). Since the points in the neighbor graph are sampled from the demonstration trajectories, the actions recorded during demonstration do not necessarily correspond exactly to those points. Instead, actions are computed as the vector from the current neighbor point to its subsequent point in the same trajectory. As a matter of notation, we represent an example trajectory point as t_i and its successor as t_i^+. Thus, given N neighbors of a plan point p, their weights w_n, normalization factor W, and the subsequent plan point p^+ are calculated as:

$$w_n = |t_n - p|$$

$$W = \sum_{n=0}^{N} w_n$$

$$p^+ = p + \sum_{n=0}^{N} \frac{w_n}{W} \left(t_n^+ - t_n\right)$$

The next step is to select a new set of neighbors. These are not merely the closest points to p^+, but must also represent portions of the demonstration trajectories at semantically equivalent states during execution of the task. Since distinct portions of the trajectories may appear in close proximity in non-Markovian regions (c.f. Figure 1) we need to rely on points that are close geodesically in the neighbor graph. However, because the demonstration trajectories may have small imperfections, rather than simply advancing to the set of points subsequent to the previous neighbors, we search

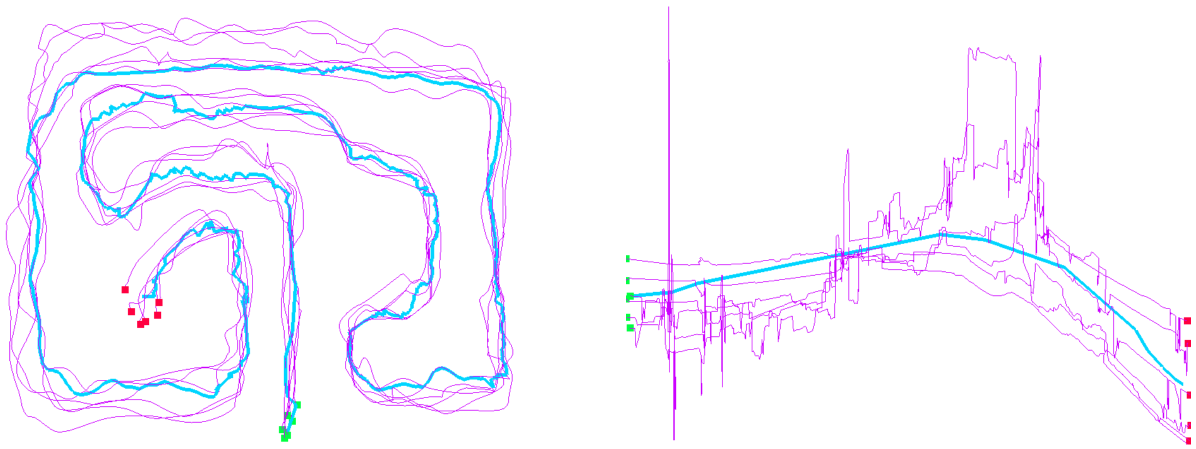


Fig. 3. Traces of the end-effector position while traversing the wire maze, shown in the workspace (left) and in the learned reduced dimensionality space (right). The light blue trace is a naïve plan created in the reduced dimensionality space, then lifted to the original workspace.

near p^+, removing neighbors that are too far away and adding new neighbors that are nearby in both metric and graph distance (Fig. 4(c)). In our experiments, we found that searching for points within d_u (the trajectory sampling distance) of p^+ and three graph links from the previous neighbor set gave good results.

Finally, we refine our choice of p^+. While the action computed from the neighbor set faithfully reproduces the actions demonstrated by the teacher at that point in the task, undesired actions are represented as well. We thus adjust p^+ toward a position that is both representative of the average action performed by the teacher and close to the new neighbor set. To do this, we place a Gaussian over each new neighbor and use gradient ascent to refine the position of p^+. Figure 4(d) illustrates level sets of this reward function around the new neighbors. The new plan point is refined to lie at a local peak of this function.

This approach to planning ensures that interpolation occurs only between demonstrated points that are similar in pose and graph distance: precisely where interpolation is expected to safely respect both geometric and non-geometric constraints. In addition, we use a volumetric model of the robot to ensure safe operation. A CAD model of the robot is used to calculate the volume of space that the robot occupies as it moves through the demonstration trajectories. These swaths of space are known to be free of physical obstacles. Then, during planning, we check whether the robot stays within the union of the swaths all along the path. If not, we use a signed distance field (SDF) [26] to further adjust the position of the planned point p^+. An SDF is an occupancy grid that, in addition to a bit indicating the safety of a given grid cell or voxel, provides a vector pointing to the nearest safe cell. This provides a computationally efficient means for adjusting trajectory points that stray into unsafe regions.

An important concern when planning is *bifurcations* in the neighbor graph: areas where demonstration trajectories diverge into two (or more) qualitatively distinct strategies. A common cause of bifurcations is physical obstacles that the teacher must circumvent. In a two-dimensional workspace, the potential for such obstacles may be reliably detected by examining the swaths of workspace occupied during training. More simply, the teacher may be instructed to demonstrate only trajectories that follow the same path around and between obstacles. If this instruction is observed, we can be assured that interpolating between any demonstrations must be safe (at least with respect to geometric constraints). All such demonstrations are *homotopic* because there exists a safe, continuous deformation between any pair of paths while keeping start and end points fixed [27]. If two demonstrations take different paths around an obstacle, say one to the left and the other to the right, a continuous deformation between them would pass through the obstacle, which is clearly not desirable.

This strategy is demonstrated in the plane by [20], but does not extend to higher dimensions. For instance, if the previous example were extended to three dimensions, there may be a safe path above the obstacle. Thus, the paths to the left and the right could safely be continuously deformed through this third path. The paths to the left and the right are thus homotopic, but a simple linear interpolation is not safe. In fact, the interpolation between them may be arbitrarily complex. A robot learner should ideally reach the same conclusion suggested by our intuition: these paths follow separate strategies, and should be treated separately. Moreover, the learner should take advantage of the teacher's knowledge in order to determine whether one strategy should be preferred over the other.

Using the SDF to detect bifurcations is a tempting, but incomplete, approach. We might produce trial robot poses by interpolating between two trajectories, then test whether these poses are safe. However, this will detect only those bifurcations caused by physical obstacles. For a more general approach, we once again rely up the neighbor graph. A bifurcation in demonstration strategies should appear as a localized partition in the graph. Example trajectories that are linked in the graph at some point in time will no longer be

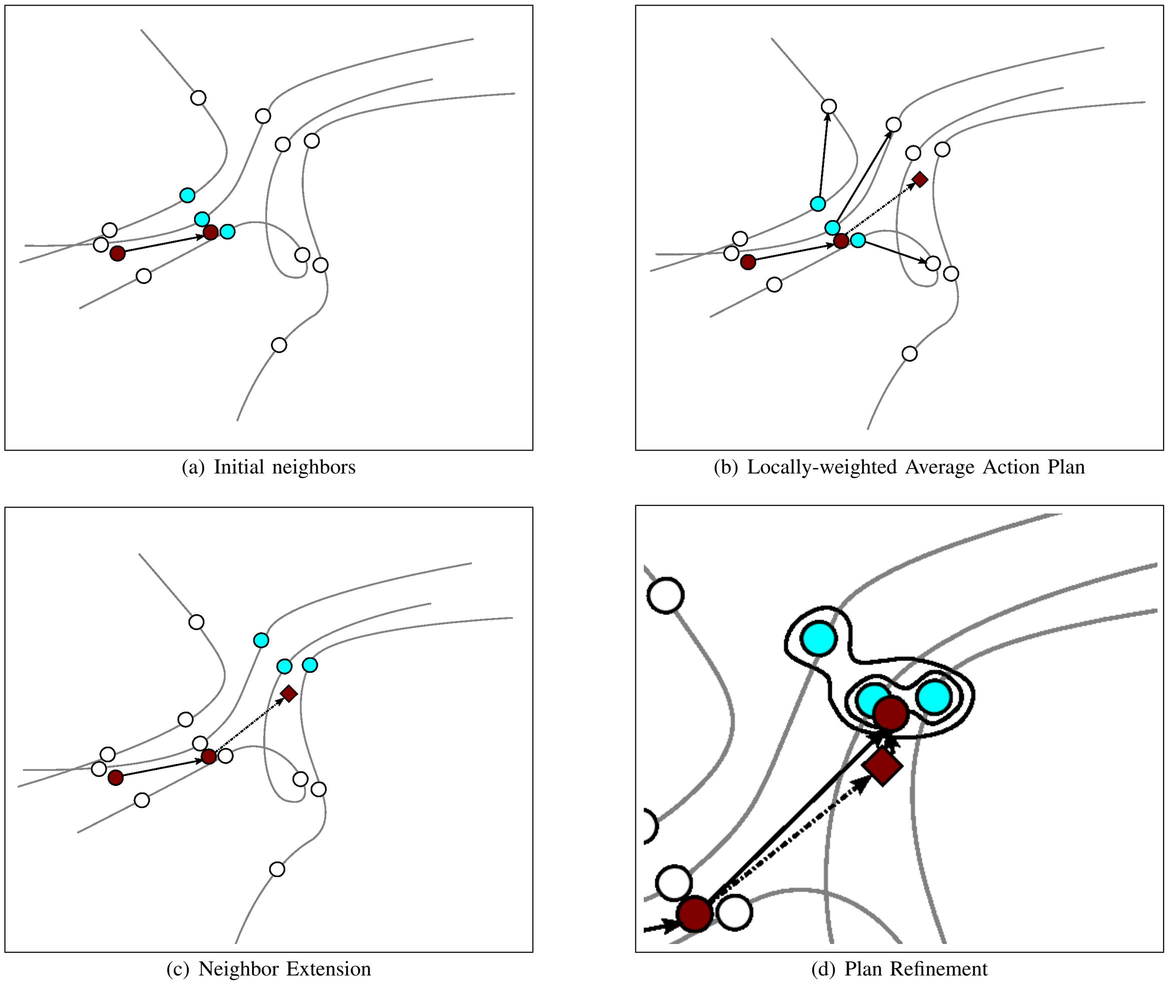

(a) Initial neighbors (b) Locally-weighted Average Action Plan (c) Neighbor Extension (d) Plan Refinement

Fig. 4. (a) The graph-based planning algorithm begins with a partial plan (red points) and a set of neighbors (blue points) selected from among demonstration trajectories (blue and white). (b) The initial estimate for the next plan point (red diamond) is computed using the locally-weighted average of neighbor actions (black arrows). (c) Neighbors (blue) are selected for the new plan point. (d) The pose of the new plan point p^+ (red circle) is refined using the new set of neighbors.

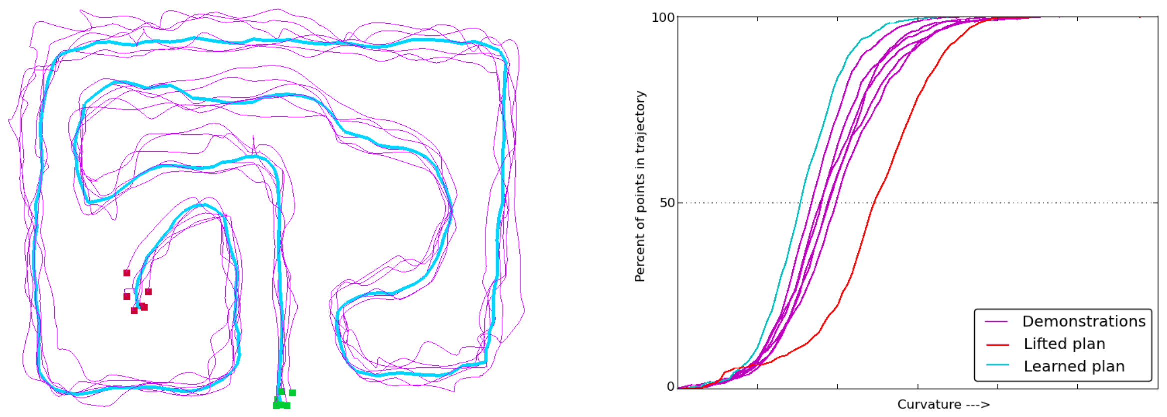


Fig. 5. The example trajectories (purple) from figure 3 with a new plan in blue. The graph on the right shows a cumulative curvature plot, illustrating discontinuities of the lifted plan (in red), while the new plan (in blue) is smoother than the examples.

linked after the bifurcation. We use the normalized cuts graph partitioning algorithm [28] to detect these locations. This algorithm compares the number of links within a partition, or cluster, to the number of links between clusters. The links between clusters are *cut* to form partitions. A good partition should cut relatively few links, with a large number of links remaining within clusters.

To apply this algorithm to trajectories, we iterate over the points of each example trajectory to find clusters of points that are densely linked to other examples relative to some point in the near future. In our experiments, we compared the interconnections at each query point to the interconnections that exist between ten and 20 steps in the future. Since cuts must occur between trajectories, all the points along a single trajectory are coalesced into a single graph node and the best normalized cut partition is calculated for each such node. In most cases, when no bifurcation exists, the normalized cuts algorithm will successfully assign all trajectories to the same cluster. Otherwise, a high score will indicate that a partition may exist. When many high scores occur within a few graph links of one another, this collection of points is considered to be a *cut neighborhood*, a region in which a bifurcation is likely to exist.

To handle bifurcations, we use *active learning*, where the robot learner asks the human teacher which branch of the bifurcation is to be preferred. Rather than present graphical representations of the trajectories on a computer screen, which may be hard for novices to interpret, the robot learner instead executes partial *trial trajectories* to demonstrate the branches of the bifurcation. In this way, the user is given a clear conception of what the robot plans to do. The user may indicate that one plan or the other is to be preferred, and future plans will follow that branch of the bifurcation. That is, when a future plan encounters points from the *cut neighborhood*, points from the preferred branch are used as neighbors, but points from the other branch are not. The user may also indicate "no preference," in which case the planner will choose a branch randomly, weighted by the number of demonstrations provided in each branch.

A final possibility is that the user discerns no distinction between the two branches. This may occur if the provided trajectories are widely separated, causing the robot to discern a bifurcation where none actually exists. In such cases, the robot requests a new example demonstrating the possibility of planning in the (currently) unknown region. The learner solicits this additional demonstration by creating a plan that approaches the bifurcation and proceeds through it, averaging the trial trajectories, until it reaches a point no longer known to be safe. At this point, the user is asked to continue the task, essentially demonstrating the equivalence of the two strategies by providing rajectory that bridges the gap. If the learner continues to detect bifurcations, additional examples are requested.

V. Experimental Results

This planning algorithm has been tested in two experimental domains. The wire maze domain (Figure 2) provides a testbed for comparison with plans created in the reduced-dimensionality latent space. Figure 5 illustrates a plan in the maze domain created by our current approach. This plan compares favorably to the jagged lifted plan of Figure 3, produced by our previous approach [1]. To quantify our claim that the current planning approach produces smooth plans, Figure 5, right, is a cumulative curvature plot that shows the percentage of points with curvature less than or equal to the value on the horizontal axis. The red curve, representing the original plan lifted from the latent space, is shifted to the right of the example trajectories, indicating that more points in this trajectory have higher curvature. The blue line, representing the new approach, is to the left of the example trajectories, indicating that it is smoother than all of the examples. This is mainly because our new approach averages out the imperfections in the demonstration trajectories. While not an explicit objective of the planner, this smoothing also decrease trajectory length. On average, in this domain, the new approach produces plans that are 1.5% shorter than the demonstrations.

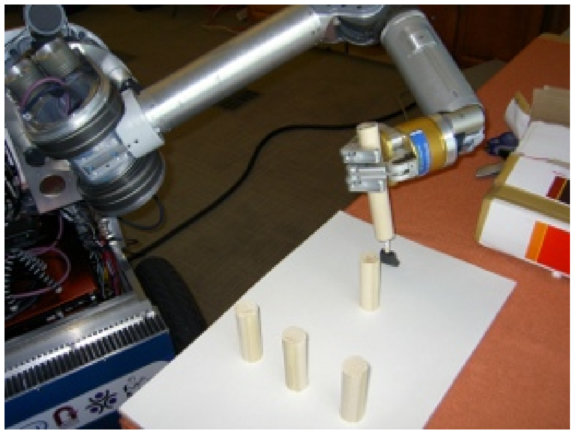


Fig. 6. The artist domain, in which the task is to sweep a paint brush across the board, while avoiding obstacles.

We also investigated the effects of adding more demonstrations. An untrained user provided six demonstrations of the wire maze task (mean curvature 3.86). With just two examples, the planner produced a plan with mean curvature of 3.86. Adding in one extra trajectory at a time, the mean curvature decreased monotonically (and approximately linearly), reaching 3.2 using all six demonstration trajectories.

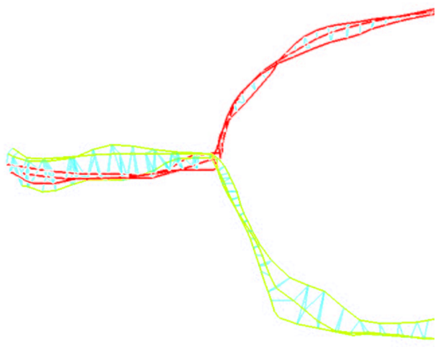


Fig. 7. A detailed view of the neighbor graph at a bifurcation.

In the second domain, which we call the *artist* domain (Figure 6), the user is asked to paint a line across a board with

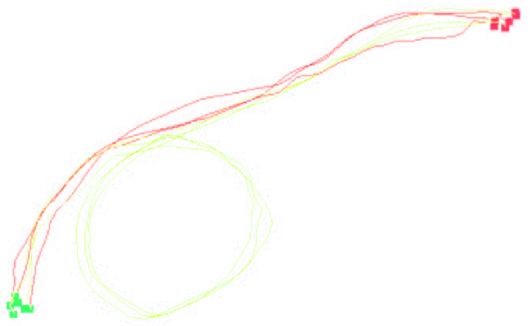


Fig. 8. Example trajectories in the artist domain. Half the examples loop around an obstacle.

vertical dowel rods protruding. This domain was developed to investigate the detection of bifurcations and the strategy for planning through them. In particular, multiple strategies are possible for traversing between and around the dowels. Two sets of example trajectories are shown in Figures 7 and 8. Figure 7 shows six demonstrations that split in half. The trajectories are shown in yellow and red, indicating their partitioning into two groups at the bifurcation, and their neighbor links are shown in blue. The six examples in Figure 8 follow similar paths across the space, but half of them take a detour in the form of a loop around one of the dowel rods. This optional detour is correctly detected as a bifurcation. Depending on the branch preferred by the user, the learner's future plans may, or may not, include this loop.

A user study was run to evaluate the ease and efficiency of training, and to test the bifurcation detection technique. 18 participants, with varying degrees of experience with robots, were recruited from Carnegie Mellon. 11 had no prior experience with robots, 4 had general experience with robots, and 3 considered themselves experienced with robotic manipulation. They were requested to provide three demonstrations each of two different strategies for achieving the task. Most complied, but one provided only a single strategy and one demonstrated four (Figure 9). The system successfully detected the bifurcations, except in the case where there was only a single trajectory for the strategy. At the end, the users filled out a survey on their experience, consisting of 13 questions on a 5 point Likert scale. Chronbach's alpha was used to cluster responses that were internally consistent, and we ended up with four general categories: ease of programming, quality of plans, training questions (the robot's active learning) and effectiveness of learning. With respect to ease of programming, all users felt that the approach was effective (mean 1.22) but inexperienced users rated the ease of use significantly higher than more experienced users. In terms of quality of plans, the users rated the plans as reasonably good (mean 1.861), although the 3 users with manipulator experience rated the plans worse (2.5, but that did not reach significance). The responses to the adequacy of the robot's training questions tended towards neutral (2.5), but with large variation, and the effectiveness

Dataset	Path Length Mean	Path Length Std. Dev.	Curvature Mean	Curvature Median
maze:				
demonstrations	1.52	0.053	3.83	3.75
planned	1.50	0.060	3.20	3.16
artist:				
demonstrations	0.702	0.211	3.64	2.83
planned	0.608	0.154	1.82	1.43

TABLE I

USER TRIAL STATISTICS

of learning question had a mean of 1.611, indicating general agreement that the robot's plan followed the strategy the teacher chose to convey. The data show a trend towards more positive assessment as experience decreases, but not to statistical significance.

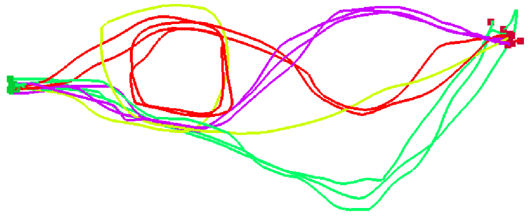


Fig. 9. A set of demonstrations from a single user. Four distinct strategies are shown in different colors. Our system detected two bifurcations, but did not detect the singleton strategy shown in yellow.

In addition, we quantified the improvements in path length and smoothness achieved by the planner in both domains. Table I presents the mean and standard deviation of path lengths across all demonstration and planned trajectories. For curvature, we instead report mean and median; standard deviation is less informative because the trajectories consist of curved and straight segments. In both cases, the planned trajectories are significantly smoother. The improvement is most pronounced in the artist domain, where the task and workspace were far less constrained, and the mean curvature is reduced by 50%. This helps support our claim that the approach is applicable to users who are not robotics experts. Similarly, the planned artist trajectories are, on average, 13% shorter than the average of the demonstrations, although there is wide variance amongst the different strategies (ranging from under 1% to 28% shorter).

VI. Conclusions and Future Work

The planning by demonstration manipulator motion planning algorithm described here is able to successfully create safe, smooth, novel plans using only demonstrations provided by a domain expert and a volumetric model of the robot. This represents a feasible method for novices in robotics to train a robot to perform sophisticated motion tasks without programming or modeling the environment. In particular, the approach enables the teacher to indicate both geometric and non-geometric constraints to the robot learner. While the approach smooths out jitter and inconsistencies, it maintains the essential characteristics of the demonstrations, including small perturbations that are consistent across the various demonstration trajectories.

One extension is to consider additional objectives, such as path smoothness constraints or non-uniform costs over regions of the work space. The current implementation preserves high-frequency noise only in areas where that noise is correlated between multiple examples. Otherwise, it is eliminated as unintended jitter when, in fact, small random motions may be necessary for some sorts of tasks. Another extension is to apply our approach to other types of robotic platforms. Although our focus has been on redundant robot manipulators, the approach is equally applicable to mobile robot platforms. Planning on a non-holonomic base is likely to present a unique set of challenges, however, particularly if dynamics are to be considered. A significant extension is to learn safe trajectories in the presence of moving obstacles. While this would necessitate detecting obstacles, we still would not have to model them explicitly within the planner. Finally, we would like to analyze our approach more formally – the approach uses a number of heuristics and parameters, and it would be useful to investigate how these choices affect performance and the guarantees associated with the approach.

In summary, we believe that our approach will enable domain experts to teach robots effectively and efficiently, with minimal training in robotics. As a result of this, and similar efforts, we anticipate that autonomous robots will become applicable to a much wider variety of domains.

VII. Acknowledgments

This work was supported by AFOSR grant #FA2386-10-1-4138 and NASA award NNX06AD23G under subcontract Z627402.

References

[1] N. A. Melchior and R. Simmons, "Dimensionality reduction for trajectory learning from demonstration," in *Proceedings International Conference on Robotics and Automation*, May 2010.

[2] H. Asada and H. Izumi, "Automatic program generation from teaching data for the hybrid control of robots," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 166–173, 1989.

[3] H. Asada, "Teaching and learning of compliance using neural nets: representation and generation of nonlinear compliance," in *Proceedings International Conference on Robotics and Automation*, vol. 2, 1990, pp. 1237–1244.

[4] J. Chen and A. Zelinsky, "Programing by demonstration: Coping with suboptimal teaching actions," *International Journal of Robotics Research*, vol. 22, no. 5, pp. 299–319, 2003.

[5] M. Stolle and C. Atkeson, "Policies based on trajectory libraries," in *Proceedings International Conference on Robotics and Automation*, 2006, pp. 3344–3349.

[6] J. Aleotti, S. Caselli, and G. Maccherozzi, "Trajectory reconstruction with nurbs curves for robot programming by demonstration," in *Proceedings International Symposium on Computational Intelligence in Robotics and Automation*, 2005, pp. 73–78.

[7] H. Friedrich, J. Holle, and R. Dillmann, "Interactive generation of flexible robot programs," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 1998, pp. 538–543.

[8] D. Bentivegna and C. Atkeson, "Learning from observation using primitives," in *Proceedings International Conference on Robotics and Automation*, vol. 2, 2001, pp. 1988– 1993 vol.2.

[9] G. Hovland, P. Sikka, and B. McCarragher, "Skill acquisition from human demonstration using a hidden markov model," in *Proceedings International Conference on Robotics and Automation*, Minneapolis, MN, 1996, pp. 2706–2711.

[10] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical system modulation for robot learning via kinesthetic demonstrations," *IEEE Transactions on Robotics*, 2008.

[11] J. D. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proceedings International Conference on Robotics and Automation*, May 2001.

[12] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in *Proceedings National Conference on Artificial Intelligence*, 1990, pp. 796–802.

[13] B. Argall, B. Browning, and M. Veloso, "Learning by demonstration with critique from a human teacher," in *Proceedings International Conference on Human-Robot Interaction*. Arlington, Virginia, USA: ACM Press, 2007, pp. 57–64.

[14] S. Chernova and M. Veloso, "Tree-based policy learning in continuous domains through teaching by demonstration," in *Modeling Others from Observations: Papers from the AAAI Workshop*, G. Kaminka, D. Pynadath, and C. Geib, Eds. American Association for Artificial Intelligence, 2006, pp. 24–31.

[15] N. Ratliff, D. Bradley, J. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press, 2007.

[16] A. Ude, C. Atkeson, and M. Riley, "Planning of joint trajectories for humanoid robots using b-spline wavelets," in *Proceedings International Conference on Robotics and Automation*, vol. 3, 2000, pp. 2223–2228 vol.3.

[17] C. Lee, "A phase space spline smoother for fitting trajectories," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, pp. 346–356, 2004.

[18] H. Mayer, I. Nagy, A. Knoll, E. Braun, R. Lange, and R. Bauernschmitt, "Adaptive control for human-robot skilltransfer: Trajectory planning based on fluid dynamics," in *Proceedings International Conference on Robotics and Automation*, Rome, Italy, 2007.

[19] E. Drumwright, O. Jenkins, and M. Matarić, "Exemplar-based primitives for humanoid movement classification and control," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 2004, pp. 140–145.

[20] N. Delson and H. West, "Robot programming by human demonstration: adaptation and inconsistency in constrained motion," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 1996, pp. 30–36.

[21] ——, "Robot programming by human demonstration: the use of human variation in identifying obstacle free trajectories," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 1994, pp. 564–571.

[22] S. Calinon and A. Billard, "Recognition and reproduction of gestures using a probabilistic framework combining PCA, ICA and HMM," in *Proceedings International Conference on Machine Learning*, 2005, pp. 105–112.

[23] O. C. Jenkins and M. J. Matarić, "A spatio-temporal extension to isomap nonlinear dimension reduction," in *Proceedings International Conference on Machine Learning*. Banff, Alberta, Canada: ACM Press, 2004, p. 56.

[24] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, Dec. 2000.

[25] A. Tsoli and O. C. Jenkins, "Neighborhood denoising for learning high-dimensional grasping manifolds," in *International Conference on Intelligent Robots and Systems*, Nice, France, Sep 2008, pp. 3680–3685.

[26] J. A. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[27] J. R. Munkres, *Topology*. Upper Saddle River, NJ: Prentice Hall, 2000.

[28] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *Proceedings Conference on Computer Vision and Pattern Recognition (CVPR)*. Washington, DC, USA: IEEE Computer Society, 1997.

Motion Interference Detection in Mobile Robots

Juan Pablo Mendoza
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213–3890
jpmendoza@ri.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213–3890
mmv@cs.cmu.edu

Reid Simmons
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213–3890
reids@cs.cmu.edu

***Abstract*—As mobile robots become better equipped to autonomously navigate in human-populated environments, they need to become able to recognize internal and external factors that may interfere with successful motion execution. Even when these robots are equipped with appropriate obstacle avoidance algorithms, collisions and other forms of motion interference might be inevitable: there may be obstacles in the environment that are invisible to the robot's sensors, or there may be people who could interfere with the robot's motion. We present a Hidden Markov Model-based model for detecting such events in mobile robots that do not include special sensors for specific motion interference. We identify the robot observable sensory data and model the states of the robot. Our algorithm is motivated and implemented on an omnidirectional mobile service robot equipped with a depth-camera. Our experiments show that our algorithm can detect over 90% of motion interference events while avoiding false positive detections.**

I. Introduction

Autonomous mobile robots have reached the point where they can start to perform useful tasks in unconstrained human-populated environments. For robots to become an efficient means for performing such tasks, they need to be able to navigate safely and effectively without supervision. Therefore, robustness during navigation (and, more generally, during task execution) is an area of increasing importance in robotics. To perform tasks robustly in unconstrained and uncertain environments, robots need to reason about their own state and execution, which is why *execution monitoring* –the problem of recognizing and indicating anomalies in behavior– has gained importance in the robotics community [1]. This paper presents a *Hidden Markov Model* (HMM)-based model for detection of *Motion Interference* (*MI*) events–events that disrupt the normal motion of the robot–with the purpose of increasing the robustness, and thus the autonomy, of robots in human-populated environments.

Previous work on execution monitoring for mobile robots has mostly focused on *model-based* monitors, such as the hierarchical monitor of Xavier [2], the knowledge-based SKEMon monitor [3], and the Unit Circle qualitative representation-based monitor [4], in all of which properties such as the dynamics of the mobile robot are explicitly modeled. More recently, there has also been robotics work in *model-free* monitors [5], in which fault detection arises solely from observing the robot's behavior, rather than from a predictive model. The model presented in this paper more closely resembles some estimation-based methods (e.g., using HMMs or Kalman Filters) that have been used for execution monitoring of outdoor robots [6], [7] and planetary rovers [8], [9]. In contrast to these methods, however, where the dynamics or the parameters of the robot are explicitly given, our model is built as an observer outside of the robot's existing architecture: the monitor builds a model from navigation data, such as driving commands and measurable velocity, as opposed to using pre-existing knowledge of the robot's properties. In this respect, our model takes some ideas from activity recognition work [10], [11], where the inner workings of the system to be described are unknown, yet behaviors are successfully classified. In some ways, then, our approach combines strengths from both model-based and model-free monitors: while an explicit model of the robot's behavior is built, this model is learned from observed data, which potentially allows for modeling of complex behaviors whose properties might be too difficult to define analytically.

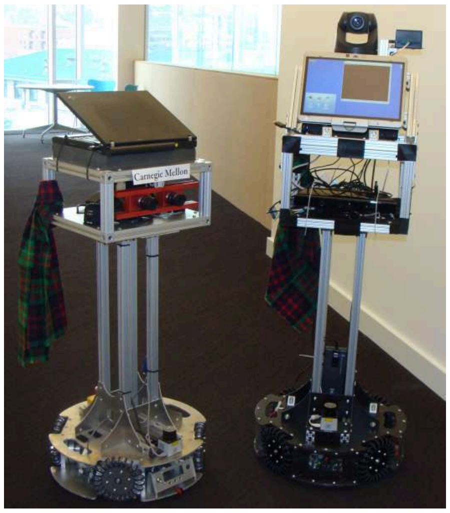


Fig. 1: The CoBot mobile service robots. CoBot robots autonomously perform tasks in human-populated environments, often without any supervision. This makes robustness during navigation a particularly relevant problem.

II. The Motion Interference detection problem

The robotic platform used to test the algorithm presented in this paper is the CoBot service robot (Figure 1). CoBots are

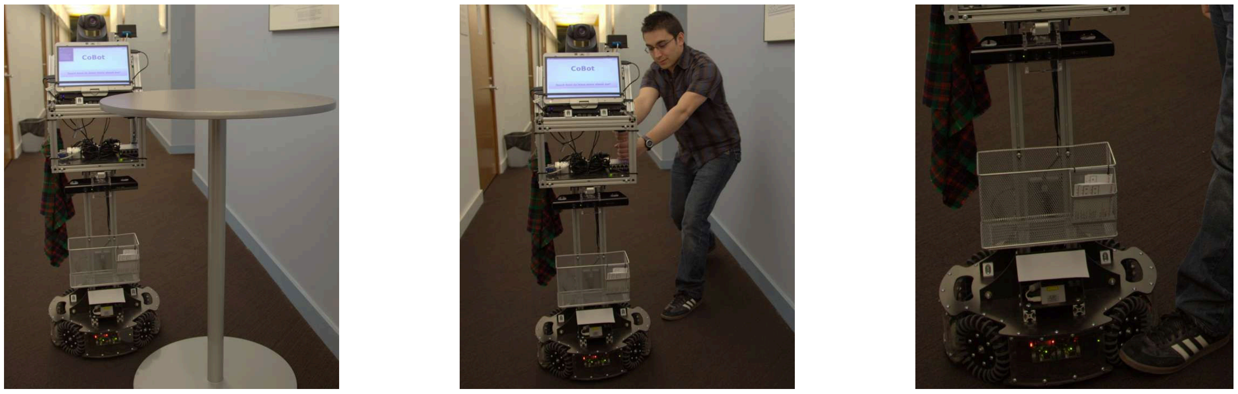


Fig. 2: Types of Motion Interference considered in this paper: (a) Collision against a partially detectable obstacle, (b) Being held by a person, and (c) Having one or more wheels stuck

equipped with an omnidirectional wheeled base for motion, laser range-finders and/or Microsoft Kinects for obstacle avoidance and localization in the environment and cameras. The level of autonomy of the CoBots is very high, having navigated more than 8.7 km while autonomously completing service tasks requested by inhabitants of the Gates-Hillman Center at Carnegie Mellon University [12]. However, the CoBots' ability to reason about their own state and performance during execution is still limited. The goal of this paper is to provide a model to increase the robustness of CoBots' (and, more generally, mobile robots') autonomous navigation by detecting when some environmental or internal event is interfering with its regular motion. For the remainder of this paper, we focus on the following types of motion interference, as they are the most relevant to the CoBot's execution, though we believe our algorithm is more generally applicable to motion interference.

Collision with partially detectable objects (*collision*)
While the fairly reliable perception mechanisms of the CoBots, combined with obstacle avoidance algorithms, can provide successful local navigation the vast majority of the time, there are some obstacles that are either undetectable or only partially detectable to the sensors. Transparent obstacles are particularly challenging for such light-based sensors, but opaque obstacles can also be only partially detectable. For example, the top of the table shown in Figure 2a is too tall for either the Kinect or the laser range-finder to perceive it. While the robot avoids the leg of the table successfully, it cannot detect and avoid the much larger full width of the table, leading to potential collisions.

Being held by a person (*hold*)
In some situations, a person may want to stop the CoBot's motion, and although the CoBots have emergency stop buttons, a person's first reaction may reasonably be to directly stop it by holding it, as shown in Figure 2b. Furthermore, the effect of being suddenly grabbed and stopped seems similar to the effect of a head-on collision between the robot and a transparent or otherwise undetectable wall. In either case, it is important for the robot to be able to detect the situation and react accordingly.

Having one or more wheels stuck (*stuck*)
Several events in the environment might cause one or more of the robot's wheels to get stuck. For example, a person might accidentally place their foot in front of the robot's wheels during navigation (see Figure 2c), or the robot could have trouble passing over gaps or level changes, such as the entrance into an elevator. A similar type of motion interference might occur independently of the environment due to malfunctioning wheel motors.

Note that all of the described forms of interference will have a similar effect on the robot: the robot's motion in the direction of travel will be impeded, and thus its velocity is going to be diminished to a value smaller than the given velocity command. Because of this similarity, all of these events are grouped for this paper under the category of Motion Interference, and are treated as equivalent events. Experimental results in Section IV support this abstraction. However, in the case of non-equivalent events (e.g., a person pushing the robot forward, or a defective motor given only a fraction of the expected current), the model can readily support detection of different types of events by adding the appropriate states to the model of Section III.

III. A HMM for Motion Interference detection

Hidden Markov Models are particularly well-suited for modeling an *MI* detector: even though the CoBots don't have sensors to directly detect *MI* events, the occurrence of these events can be inferred from the observations that are available to the robot. HMMs provide an appropriate framework to perform these inferences.

A Hidden Markov Model M can be defined as a 5-tuple $M = \{S, \Pi, A, O, B\}$, where

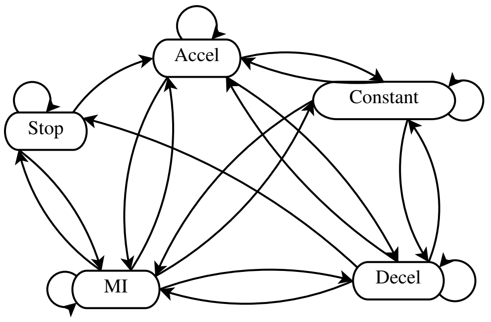


Fig. 3: Diagram of the HMM modeling the Motion Interference Monitor. Ellipses represent hidden states, while arrows represent transitions between states. (While no arrows are shown in transitions which the data determined had probability 0, no transitions were explicitly forbidden)

$S = \{s_i\}$	the set of hidden states in M
$\Pi = \{\pi_i\}$	the initial distribution of M such that $P(S_1 = s_i) = \pi_i$
$A = \{a_{ij}\}$	the transition probabilities such that $P(S_{t+1} = s_j \mid S_t = s_i) = a_{ij}$
$O = \{\vec{o}_i\}$	the space of possible observations
$B = \{b_i\}$	the emission probabilities such that $P(\vec{o}_j \mid S_t = s_i) = b_i(\vec{o}_j)$

For the detector described in this paper, each possible simple behavior of the robot is represented by a state in S. The possible behaviors assigned to the CoBot for purposes of *MI* detection are $S = \{stop, accel, constant, decel, mi\}$ representing the states where the robot is stopped, accelerating, at constant positive speed, decelerating, and having its motion interfered, respectively. A diagram for the resulting HMM is shown in Figure 3. Since the robot always starts from the *stop* state, the respective values for Π are $\Pi = \{1, 0, 0, 0, 0\}$.

True transition probabilities between pairs of states were approximated from a large amount of hand-labeled gathered training data. In general, given a set of observations accompanied by state labels S_t for all times $0 \leq t \leq T$,

$$a_{ij} = \frac{\sum_{t=0}^{T-1} \left(\delta_{S_t,s_i} \cdot \delta_{S_{t+1},s_j} \right)}{\sum_{t=0}^{T-1} \delta_{S_t,s_i}}, \tag{1}$$

where $\delta_{i,j}$ is the Kronecker delta function. That is, the transition probability from state s_i to s_j is given by the total number of labeled observations in which the robot transitioned from state s_i to s_j divided by the total number of labeled observations in which the robot transitioned from s_i to anywhere. The only transition probabilities not calculated using Equation 1 were transitions into *MI* $a_{i,mi} \forall i \neq mi$. Given the rarity of *MI* events in normal CoBot runs, *MI* events had to be artificially created to gather significant data for experimenting, and thus the real probabilities to transition from a different state to the *MI* state are extremely difficult to gather. These values were thus set to $a_{i,mi} = p_{mi}$, where p_{mi} is the only parameter for the *MI* detector. Tuning p_{mi} has the effect of varying the total number of *MI* events detected, and therefore it serves as a parameter to find the desired trade-off value between precision and recall of *MI* events (see Section IV for more details).

Observation space O may vary greatly depending on the sensors of the specific robot and the type of event that needs to be detected. For the task of detecting *MI* events in the CoBot, observations were of the form

$$\vec{o}_t = (u_t - v_t, a_t, j_t, u_t), \tag{2}$$

where u_t is the commanded forward velocity of the robot, and v_t, a_t and j_t are the forward velocity, acceleration and jerk of the robot as measured by its wheel encoders. The reasoning for each observation and the method for obtaining it are the following:

Velocity difference $\mathbf{u_t - v_t}$.
One can expect that when the robot's movement is being interfered, its measured velocity would be significantly lower than its commanded velocity. While u_t is directly obtained as a command, v_t needs to be calculated from the individual encoder velocities. Velocity v_t was obtained from the least squares solution transforming the encoder values for each of the four wheels to forward, sideways and rotational components of the robot's velocity.

Acceleration $\mathbf{a_t}$.
There are times during normal (i.e., not *MI*) robot motion when the velocity difference $u_t - v_t$ is significantly positive. For example, when the robot accelerates from a stopped position to full speed, the change in u_t is discrete, while v_t smoothly changes from 0 to u_t over time. Thus, $u_t - v_t$ alone is not a sufficient observation to detect *MI* events. Therefore, acceleration a_t is added as an additional layer to distinguish between these events: while an accelerating robot has a positive acceleration, a robot during a *MI* event usually has either negative (at the moment the *MI* event begins) or near-zero (once velocity has stabilized) acceleration. a_t can be obtained by applying linear regression to the last N_a measures of velocity v_t as a function of t, and then getting the slope of the resulting line. Even though a_t could be obtained from only the last 2 values of v_t and t, a larger number N_a is used to reduce the effects of noise in the data. N_a must be large enough to negate the effects of noise, but small enough to provide a meaningfully recent value for a_t. For this paper, $N_a = 4$.

Jerk $\mathbf{j_t}$.
In the event that the robot's motion is disrupted while the robot is accelerating, it might be the case that the positive acceleration continues for a while until the final velocity under the disturbance is reached. In these cases, a_t might be within the normal parameters of an accelerating motion at any instant, but the change in acceleration in time may provide valuable information to distinguish *MI* acceleration from normal acceleration. For this

reason, the second time derivative of the velocity is also used as part of the observations. Jerk j_t is obtained by dividing the difference between the two last acceleration measurements by the difference in time between measurements. However, since jerk is significantly more sensitive to noise in the data than acceleration, a larger number of measurements $N_j \geq N_a$ is used to calculate the accelerations to be used for j_t, for purposes of further smoothing of noise at the cost of a more outdated jerk measurement. For this paper, $N_j = 8$. Various values of N_a and N_j were tested on small portions of the data, and the final values were those that maximized performance in these portions.

Velocity command $\mathbf{u_t}$.

The final attribute of the observation is simply the commanded velocity u_t. The purpose of this attribute is to distinguish between *stop* , where $u_t = 0$, and *constant*, where $u_t \neq 0$. Otherwise these states would have identical properties.

There are some design decisions behind the use of the attributes of Equation 2 as observations. First, notice that only the forward velocity, acceleration and jerk of the robot are used. While the CoBot's base is omnidirectional, most of its movements are restricted to the forward direction (plus rotations) because its sensors are pointing forward. Therefore, using only the forward direction has the benefit of requiring estimation of fewer parameters at little cost. Furthermore while the CoBot has other sensors (e.g., Kinect) that could provide estimates of velocity apart from the encoders, for this paper only encoder estimates were used. While encoders provide the simplest method for velocity estimation, the biggest concern with using them as the only estimator is that, on extremely slippery surfaces, the robot's wheels could keep spinning at the commanded velocity even during an *MI* event. We determined, however, that even in the most slippery surfaces in the CoBots' environment (i.e., hardwood floors), slipping was not enough to make *MI* events undetectable using only encoder-based velocities. If this were not the case, however, additional sources of velocity information could be added as observations at the cost of additional parameter estimation.

Finally, emission probabilities B are calculated from hand-labeled training data in a similar manner to transition probabilities A. For the specific monitor described in this paper, the data is treated as being generated by conditionally independent Gaussian variables. The Gaussian assumption fits the data relatively well, but the independence assumption does not hold for the data in question: velocity, acceleration and jerk are strongly correlated. While the general model readily supports treating data as conditionally dependent, there is a trade-off between higher model accuracy at the cost of more parameter estimation and processing (conditionally dependent) and lower model accuracy with a simpler and more efficient model (conditionally independent). For the purposes of this paper, the simpler model provided satisfactory results (see Section IV), and therefore the it was preferred over the more complex one. Rewriting the observation attributes as $(o_1 \equiv u_t - v_t, o_2 \equiv a_t, o_3 \equiv j_t, o_4 \equiv u_t)$ for brevity, the emission probabilities can then be written as:

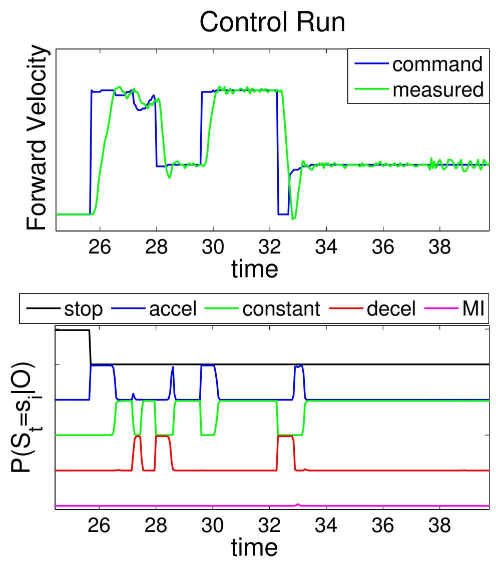


Fig. 4: Sample of data gathered from a normal (control) run of the CoBot navigating in its environment. The top figure shows the velocity command and the measured velocity (meters per second) over time (seconds)), while the bottom stacked probability figure shows the respective assigned probabilities $P(S_t = s_i|O)$ (each between 0 and 1) for each state given the sequence of observations.

$$
\begin{aligned}
b_i(o_1, o_2, o_3, o_4) &= P(o_1, o_2, o_3, o_4|S_t = s_i) \\
&= \prod_{j=1}^{4} P(o_j|S_t = s_i) \\
&= \left(\prod_{j=1}^{3} f(o_j; \mu_{i,j}, \sigma^2_{i,j})\right) P_i(o_4), \quad (3)
\end{aligned}
$$

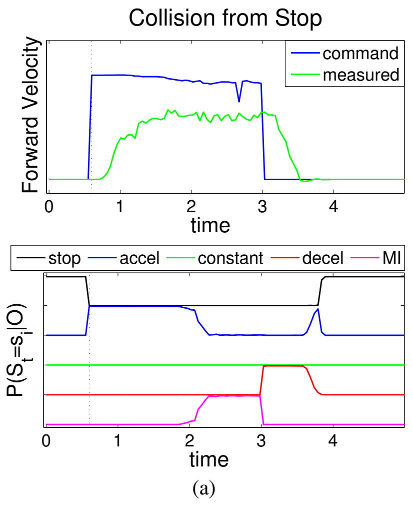
where the individual probabilities are defined as:

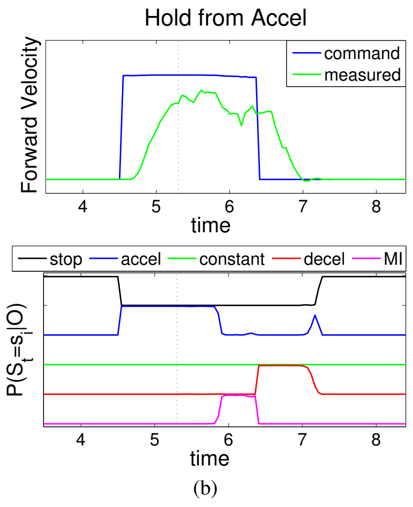
$$
f(o_j; \mu_{i,j}, \sigma^2_{i,j}) = \frac{1}{\sigma_{i,j}\sqrt{2\pi}} e^{-\frac{(o_j-\mu_{i,j})^2}{2\sigma^2_{i,j}}} \quad (4)
$$

$$
P_i(o_4) = \begin{cases} \delta_{0,o_4} & \text{if } i = 1 \\ 1 & \text{if } i = 2,4,5 \\ 1-\delta_{0,o_4} & \text{if } i = 3 \end{cases} \quad (5)
$$

Equation 4 simply describes a Gaussian probability distribution whose parameters were obtained from training labeled data, while Equation 5 describes the probability of observing a certain velocity command $u_t = o_4$ depending on the current state: in state *stop*, $o_4 = 0$ always; in state *constant*, $o_4 \neq 0$, and in any other state o_4 could be anything.

Having defined all S, Π, A, O, B, the HMM-based *MI* detector is fully defined. Now, given any sequence of observa-





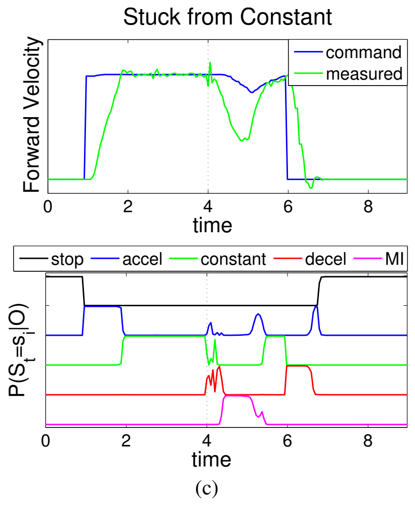


Fig. 5: Examples of data gathered from *MI* runs. As in Figure 4, top figures show velocities while bottom figures show probabilities for each state, as predicted by the monitor. Figures show (a) a collision against a partially detectable obstacle right as the robot starts to move, (b) somebody holding the robot as it is accelerating, and (c) something interfering with a wheel's rotation when the robot is traveling at constant speed. Vertical dotted lines indicate the beginning of an *MI* event, while rise in $P(S_t = s_{mi}|O)$ above 0.5 indicates detection of the event.

tions, the probability of being in state *MI* at each time can be calculated using an algorithm such as the forward algorithm described in [13]. Then, if $P(S_t = s_{mi}|O) > thresh$ (for this paper, $thresh = 0.5$), an *MI* event has been detected at time t. Changing the value of $thresh$ varies the sensitivity of the detector, but for this paper it was kept fixed, as parameter p_{mi} already served this purpose.

IV. Detector performance results

A. Methods

To gather the necessary data for training and testing of the detector, two long control runs (no *MI* events) and 29 short test runs (with *MI* events) were conducted on the CoBot robot. For each run, the robot was instructed to autonomously move from its current location to a different location in the building; the driving commands, encoder-based velocity and times of *MI* events (perceived by a human observer) were then recorded. The control runs, during which a human supervisor made sure no *MI* events happened, lasted about 3 minutes each, and gave a total of 7145 observations. A subset of the data gathered during a control run is shown in Figure 4. The test runs were each much shorter, focusing on the *MI* event, and giving a total of 8101 observations. Of the test runs, 15 contained $collision$ events, 6 contained $hold$ events, and 8 contained $stuck$ events. Figure 5 shows the data gathered from three of these experiments.

To train and test the detector, each observation was manually labeled as $stop$, $accel$, $constant$, $decel$ or *MI*. Since *MI* times were previously recorded, each observation during those periods was labeled as *MI*. When the robot travels at constant speed, the standard deviation of the encoder-measured velocity is about $\sigma = 0.028$m/s. From this, each observation where the velocity command u_t was 0 and the measured velocity v_t was within σ of 0 was labeled as $stop$. Similarly, each observation where $|v_t - u_t| \leq \sigma, u_t \neq 0$ was labeled as $constant$. To label $accel$ (or $decel$), every observation after an increase (or decrease) of u_t, and while $|v_t - u_t| > \sigma$ was labeled as $accel$ (or $decel$, respectively). Other observations (i.e., noisy observations where $|v_t - u_t| > \sigma$ but u_t had been constant) were labeled as $constant$.

The detector's performance was then tested using leave-one-out cross-validation: for a given parameter set, 31 tests were conducted (one for each labeled run of the robot). For test i, all runs except for run i were used for training the detector (i.e., finding transition and emission probabilities), which was then tested on run i. A *true positive* detection happened when at least one frame within a *MI*-labeled interval was classified as *MI*. A *false positive* detection was defined as each group of consecutive frames outside of the *MI*-labeled intervals that were classified as *MI*, given that such group was not a continuation of a true positive detection (the probability of being in state *MI* could take a few frames to decay after an *MI* event; this was not considered a false positive). A *false negative* was defined as each *MI*-labeled time interval where no *MI* event was detected.

B. Results

The goal of the detector presented in Section III is to detect *MI* events reliably and within a useful time from initial interference. The proposed measures for judging the model are therefore precision (the fraction of detected *MI* events that were true *MI* events) and recall (the fraction of true *MI* events detected) rates of detection, as well as the average and median time to detection from when interference starts.

The only parameter of the model, transition probability p_{mi}, was varied to find the trade-off between precision and

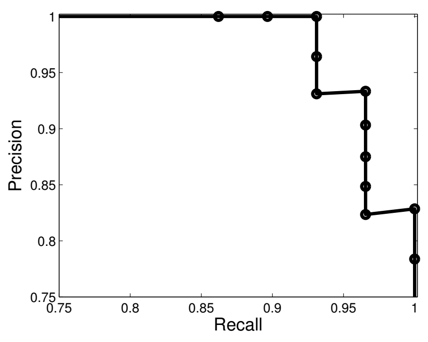


Fig. 6: Trade-off between precision and recall rates for motion interference detection, as parameter p_{mi} is varied.

recall rates. For each tested parameter value, a *full test* –i.e., a set of 31 cross-validation tests as described in Section IV-A– was conducted, with the results of Figure 6. For the purposes of the CoBot project, optimization for high precision was prioritized, given that the project focuses on giving a high degree of autonomy to the robot, and stopping execution for false positive detections would hinder this autonomy.

An optimal parameter value for our purposes was $p_{mi} = 5 \times 10^{-8}$, since it had the highest recall rate for which 0 false positives were detected. For this value, the precision rate was 100% while the recall rate was 93.1%; the average time to detection from initial motion interference was $\bar{t} = 0.647$ seconds. The median time to detection was significantly lower than this, at $t_m = 0.36$ seconds, reflecting the fact that a few outlier detections took significantly longer than the average detection time. These outliers were mostly *MI* events that started from the *stop* state (e.g., Figure 5a); this can be explained by the fact that the wheel's accelerations looked normal in the beginning, even if they were mostly slipping while spinning, before their behavior was abnormal enough to be detected as an *MI* event. This suggests that perhaps adding an estimate of velocity from a different sensor as an observation could help diminish the average time of detection. Overall, however, the detection time was usually well under a second, which is a useful time-frame for many applications, such as stopping when being held from a dangerous situation, or reacting to an inescapable collision.

V. Conclusion

We presented an HMM-based model for Motion Interference (*MI*) detection for a mobile robot. We identified the sensory observables of the robot and three types of motion interference. Through experiments conducted on the CoBot service robot, we have shown that such a model can successfully detect events that are not directly perceivable by the robot. Our work in general contributes an approach in which robots can reason about specific events by looking at their internal and external sensed state with input from their commanded controls. While this work focuses on the detection of *MI* events rather than actions to recover from them, we considered a base stop command to the robot when the event is detected, and will pursue research on other possible actions.

The *MI* events we considered limit the forward velocity of the robot, but other types of *MI* events could be detected using a similar approach (e.g., pushing the robot so that its measured velocity is above its velocity command). It is in principle feasible to detect anomalies in the behavior of the robot even if these anomalies have not been explicitly modeled: the formulation of HMMs allows us not only to calculate the probability of being in a particular state given a series of observations, but also the probability that a model describes the observable of a robot given a particular series of observations [13]; one could thus expect that a robot that has fallen in an unmodeled state would yield a significantly different model probability distribution than a robot running normally (i.e., within the model). In this way, HMM-based models for execution monitoring could provide a natural model for implementation of hybrid model-based and model-free monitoring. Finding whether this is a practical method to detect anomalies in our robots is a topic of future research.

VI. Acknowledgments

This work was partially supported by the National Science Foundation award number NSF IIS-1012733 and the AFOSR grant number FA2386-10-14138. The views and conclusions contained in this document are those of the authors only.

References

[1] O. Pettersson, “Execution monitoring in robotics: A survey,” *Robotics and Autonomous Systems*, vol. 53, no. 2, pp. 73–88, 2005.
[2] J. Fernandez and R. Simmons, “Robust execution monitoring for navigation plans,” in *Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems*, 1998.
[3] A. Bouguerra, L. Karlsson, and A. Saffiotti, “Monitoring the execution of robot plans using semantic knowledge,” *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 942–954, 2008.
[4] H. Liu and G. M. Coghill, “A model-based approach to robot fault diagnosis,” *Knowledge-Based Systems*, vol. 18, no. 4-5, pp. 225–233, 2005.
[5] O. Pettersson, L. Karlsson, and A. Saffiotti, “Model-free execution monitoring in behavior-based robotics,” *IEEE Trans. on Systems, Man and Cybernetics, Part B*, vol. 37, no. 4, pp. 890–901, 2007.
[6] S. Scheding, E. Nebot, and H. Durrant-Whyte, “The detection of faults in navigation systems: A frequency domain approach,” in *Proceedings of the Int. Conf. on Robotics and Automation*, 1998, pp. 2217–2222.
[7] E. M. Nebot, H. F. Durrant-Whyte, and S. Scheding, “Frequency domain modeling of aided GPS for vehicle navigation systems,” *Robotics and Autonomous Systems*, vol. 25, no. 1-2, pp. 73–82, 1998.
[8] R. Washington, “On-board real-time state and fault identification for rovers,” in *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2000, pp. 1175–1181.
[9] V. Verma, G. Gordon, R. Simmons, and S. Thrun, “Particle filters for rover fault diagnosis,” in *IEEE Robotics & Automation Magazine special issue on human centered robotics and dependability*, 2004.
[10] D. Govindaraju and M. Veloso, “Learning and recognizing activities in streams of video,” in *Proceedings of the AAAI Workshop on Learning in Computer Vision*, 2005.
[11] K. Han and M. Veloso, “Automated robot behavior recognition,” in *Proceedings of IJCAI Workshop on Team Behaviors and Plan Recognition*, 1999.
[12] M. Veloso et al., “Symbiotic-autonomous service robots for user-requested tasks in a multi-floor building,” 2012.
[13] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

Mobile Robot Fault Detection based on Redundant Information Statistics

Juan Pablo Mendoza
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jpmendoza@ri.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
mmv@cs.cmu.edu

Reid Simmons
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
reids@cs.cmu.edu

***Abstract*— Detecting and reacting to faults (i.e., abnormal situations) are essential skills for robots to safely and autonomously perform tasks in human-populated environments. This paper presents a fault detection algorithm that statistically monitors robot motion execution. The algorithm does not model possible motion faults, but it instead uses a model of normal execution to detect anomalies. The model of normal execution is based on comparisons between redundant sources of information; specifically, wheel encoder readings and localization algorithm output are used as the redundant sources of displacement information. The algorithm was implemented on a service robot that often navigates in a human-populated environment without supervision. Experimental results show that the algorithm can detect even minor motion faults and stop execution immediately to guarantee safety to the humans around the robot.**

I. Introduction

As autonomous mobile robots start to populate human environments, safety during execution is becoming a primary concern for researchers and developers. This rising necessity for robustness in unconstrained environments has made *execution monitoring* –the problem of recognizing and indicating anomalies in behavior– increasingly prominent in the robotics community [6]. An important feature of unconstrained, human-populated environments is their richness: in such complex environments, it is not realistic to assume that the robot's designers can foresee every way in which execution could fail. Therefore, execution monitors that explicitly require *faulty execution models* (e.g., motion interference [4] or wheel failures [10]) are helpful in detecting faults quickly, but not sufficient to detect unforeseen faults. This work, on the other hand, does not assume any previous knowledge of the ways in which execution could fail, but instead only requires some *normal execution model*, and uses this model to detect deviations from normal behavior. Once a significant deviation from normal behavior has been detected, the robot can, at the very least, stop immediately until a qualified operator can fix the problem.

For this paper, normal execution models are created using *redundant information*: equivalent information that is provided by multiple sources. For example, robot location can be provided by both Wi-Fi signal and visual landmarks. Information obtained from these multiple sources is approximately equal under normal circumstances, and significant deviations from this are symptoms of failure in execution.

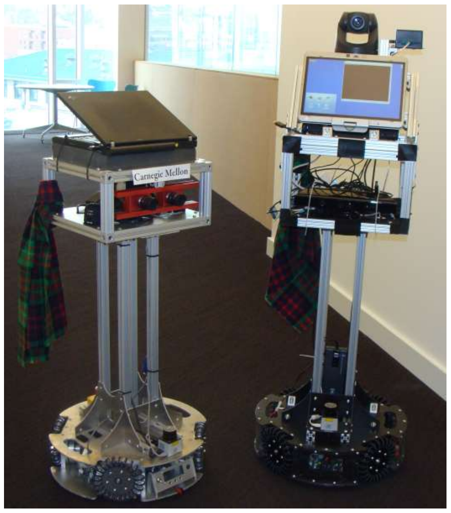


Fig. 1: The CoBot mobile service robots. CoBot robots autonomously perform tasks in human-populated environments, often without any supervision. Autonomous execution monitoring during navigation is thus essential for the safety of people around them.

The robotic platform used to test the algorithm presented in this paper is the CoBot service robot (see Figure 1). The CoBots are exposed to unsupervised physical interaction with humans constantly, having navigated among humans more than 100 km while autonomously completing service tasks requested on demand by inhabitants of the Gates-Hillman Center at Carnegie Mellon University [9]. Since the CoBots often perform these tasks without the supervision of any member of the developing team, safe autonomous interaction with humans is a high research priority. Each CoBot is equipped with an omnidirectional four-wheel base for motion (and encoders for odometry), laser range-finders or Microsoft Kinects for obstacle avoidance and localization in the environment and cameras. Under normal execution, many precautions are taken to ensure safe execution around humans (e.g., if there is no safe path around a human, the CoBots stop and ask the human to move, instead of attempting to steer around him or her). However, these algorithms may fail under abnormal circumstances, such as malfunctioning obstacle-detection sensors, leading to potentially unsafe execution. Prompt detection and recovery from abnormal situations is thus essential for robustness and safety of the CoBots.

II. Probabilistic failure detection

The monitor described in this paper operates by comparing equivalent observations (at time t) o_{t1} and o_{t2} given from two separate sources. During normal execution, the difference $x_t = o_{t1} - o_{t2}$ between them is expected to be close to 0. However, in real-world applications, with noisy sensors and actuators, this difference at each time-step is usually not exactly 0, and even in the limit as time goes to infinity, the expected difference may be close to but not exactly 0. Because of this, the approach used in this work is probabilistic in nature, incorporating uncertainty naturally into the model.

Formally, the method for detecting faults is the following: Given a set of comparison observations $X = \{x_1, x_2, \ldots, x_T\}$ with sample mean $\bar{X}$, the algorithm estimates the probability that the true mean μ of the underlying model lies within some acceptable interval $[\mu_-, \mu_+]$ around 0. That is, the algorithm calculates $P(\mu_- \leq \mu \leq \mu_+)$. To do this, one can first define a standardized variable Z:

$$Z = \frac{\bar{X} - \mu}{\sqrt{\frac{\sigma^2}{n}}}, \tag{1}$$

where σ^2 is the variance of X and n is the size of X. The standardized problem then becomes that of calculating $P(Z_- \leq Z \leq Z_+)$, where Z_- and Z_+ are calculated analogously to Z from μ_+ and μ_- respectively. Given that these variables are in standard form, the desired probability can be obtained straightforwardly from the standard cumulative normal distribution function $\Phi(z)$:

$$\begin{aligned} P(\mu_- \leq \mu \leq \mu_+) &= P(Z_- \leq Z \leq Z_+) \\ &= P(Z \leq Z_+) - P(Z < Z_-) \\ &= \Phi(Z_+) - \Phi(Z_-) \end{aligned} \tag{2}$$

After calculating this probability, detecting failures in execution is reduced to choosing a threshold probability p_{thresh} below which it is too unlikely that the observed data still fits the expected model.

III. Redundant information in the CoBot

The CoBots have several sensors and algorithms from which redundant information can be obtained to monitor their execution. The algorithm presented here can be used on any element of the following non-exhaustive list of redundant information:

Location and displacement. Several sensors and algorithms can provide information at each timestep about the robot's location or displacement. Currently, the commanded velocity, wheel encoders, laser range-finder and Microsoft Kinect are used to localize the robot using an augmented particle filter [1], [2]. Each one of these, including the output of the localization algorithm itself, can be used as a source of location or displacement information. The CoBots also have infrared detectors for identification of pre-specified landmarks in the environment, which can provide this information as well. Furthermore, streams of images from the RGB cameras could be used to obtain location or displacement information, with algorithms such as visual odometry [5] or visual landmark recognition [8], although these are not currently implemented on the CoBots.

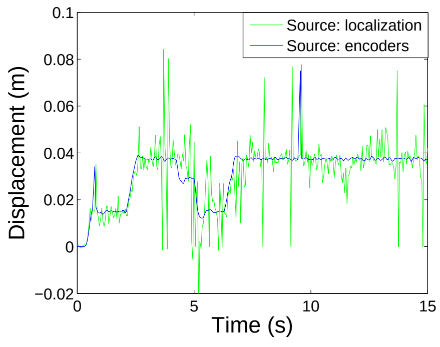
Time to task completion. The knowledge base of the CoBots include an estimate of how long each of its tasks is expected to take [3]. This knowledge can be computed both at the higher levels (e.g., how long it takes to fetch an object from a certain office) as well as the lower levels (e.g., how long it takes to traverse a particular segment in its navigation planning graph). This expected knowledge can be compared to the actual measured time of task completion to look for faults in execution.

Expected object locations. One of the CoBots' algorithms [7] for finding objects in their environment queries the internet to infer the probability that a certain object will be found in a certain space (e.g., the probability that coffee is found in the kitchen). While searching for these items in their environments, the CoBots could compare the inferred probability returned by this algorithm to the actual experienced frequency with which it finds such objects, to find abnormal situations (e.g., if someone is actively hiding all the coffee from the CoBots).

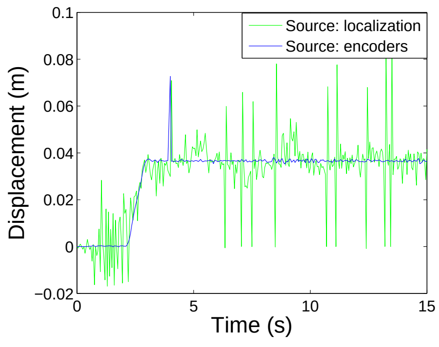
One of the most immediate safety concerns for the CoBots is the need to be confident that the robots are moving as expected while navigating among humans, without running into problems such as wheel motor or encoder malfunctions, collisions against imperceptible obstacles, getting lost, and others. Many of these motion failures can lead to dangerous situations (e.g., robots drifting to unsafe territory or attempting to drive through unperceived objects), and therefore this paper focuses on detection of motion failures. (Once a motion failure has been detected, the simplest safe action is taken: the robot stops immediately.) The monitor in this paper was thus trained to detect failures in displacement data, and the two redundant sources were the wheel encoders and the output of the localization algorithm. These sources of information were chosen because their output can be relatively simply used to obtain displacement at each timestep.

To obtain displacement (in robot x, y, and heading θ coordinates) from the wheel encoders, the displacement of each of the four encoders is mapped to x, y and θ using a least squares solution. To obtain displacement from the localization algorithm, x, y and θ displacement in world coordinates (obtained by comparing locations at consecutive timesteps) are transformed to robot coordinates by applying the appropriate rotation. Figure 2a shows the resulting normal execution displacement data as obtained from these two sources. Notice that the noise in this data is very significant, and therefore a statistical approach to fault detection is necessary.

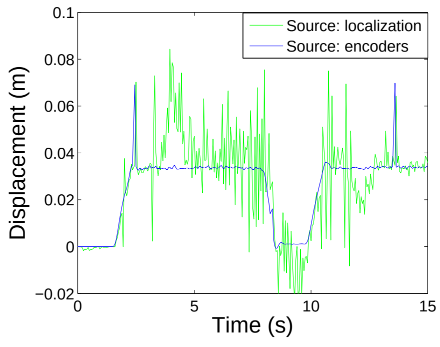
To fully define a monitor based on these observations, parameters σ^2, μ_-, μ_+ and p_{thresh} must be defined for each of



(a)



(b)



(c)

Fig. 2: Encoder and localization-based displacement data. (a) shows the first 15 seconds of data during normal execution, while (b) and (c) show similar periods with $\varepsilon = -0.1$ and $\varepsilon = -0.4$ respectively. Note that there is no obvious difference between $\varepsilon = 0$ and $\varepsilon = -0.1$ from visual inspection, but at $\varepsilon = -0.4$, the abnormal behavior is more clear

the dimensions (x, y, θ). For this paper, these parameters were chosen to be equal for each of the dimensions for simplicity, although this does not need to be the case. Also, parameter values were chosen to be conservative towards autonomy —i.e., they were chosen with the goal of minimizing false positive fault detections. For the experiments in this paper, $\sigma^2 = 0.001$ was chosen as a conservative approximation of the measured $\sigma^2 = (0.00019, 0.00005, 0.00012)$ for (x, y, θ) respectively. Similarly, $\mu_- = -0.001$ and $\mu_+ = 0.001$ were chosen as conservative approximation of the measured long-term error during normal execution (the measured value was $\mu = (-0.0007, -0.0003, -0.0004)$. Finally, $p_{thresh} = 0.01$ was chosen to be low enough so that no false positives were detected during normal execution.

IV. Experiments and results

To test the execution monitor's performance, a wheel encoder malfunction was artificially induced. During each timestep of the CoBot's navigation, the reported displacement of one of the four encoders (always the same one) deviates from the true encoder displacement reading by a pre-specified fraction ε of the true displacement. For example, $\varepsilon = 0$ represents normal execution, while $\varepsilon = -0.1$ represents a situation where three of the wheel encoders work normally, but the fourth one reports $0.9d$, where d is the true displacement of the fourth wheel. Figures 2b and 2c show the displacement data from wheel encoders as well as localization algorithm output for two values of ε. For the experiments in this paper, all values of ε where less than 0; however, preliminary tests suggest the monitor can detect faults with $\varepsilon > 0$ as well.

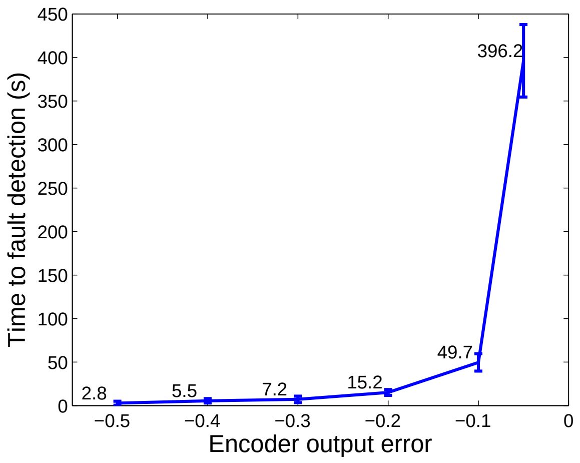
Different values of ε were tested to determine how detection times would vary as a function of how subtle the fault to be detected was. For each ε value, 10 tests were run to find the average time to detection. For each test, the robot was instructed, at a high level, to autonomously move to different destinations in the building, avoiding obstacles and interacting with people as usual. During each test, every time a new wheel encoder observation was received, $P(\mu_- \leq \mu \leq \mu_+)$ was calculated as shown in Equation (2), using all the observations received by that time. If $P(\mu_- \leq \mu \leq \mu_+)$ fell below p_{thresh}, a fault was detected. Results are shown in Figure 3. As is shown in the figure, and consistent with intuition, the time required to detect faults increases significantly as the tested fault gets more subtle and thus more observations are necessary to be confident that they were not produced by the expected model. Some faults more minor than $\varepsilon = -0.05$ (e.g., $\varepsilon = -0.01$ was tested) are not detectable by the monitor, since these subtle faults maintain the true deviation μ within the acceptable margin $[\mu_-, \mu_+]$ even as time approaches infinity.

V. Future Work

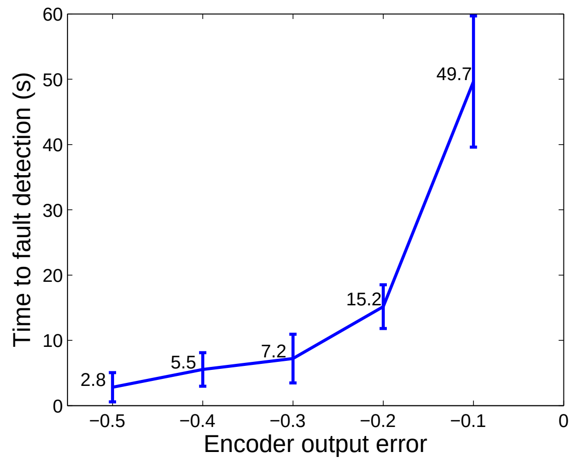
While the monitor presented in this paper has shown promising detection results for detection of one type of unmodeled failure, two directions for future work seem clear.

A first direction to expand this work is to test the monitor on several different faults. One of the key strengths of the monitor presented here is that it does not explicitly model faults, but instead it uses information redundancy to assess normality. This means that other kinds of motion failures (e.g., collisions, getting lost) should also be detectable by this monitor. Preliminary tests were conducted to test whether this monitor could detect collisions against imperceptible obstacles; on average, collisions were detected 1.98 seconds after the collision started (significantly slower than the 0.65 seconds in [4], but without explicitly modeling the fault). Showing detection of multiple kinds of unmodeled faults is essential to show the flexibility of this monitor.

As a second direction for future work, notice that one of the biggest limitations of the monitor, as presented in this paper, is that all faults are considered to happen globally. At each timestep, all the observations accumulated throughout the current run of the robot are grouped into a single statistic, which is then analyzed to determine its normality. This global assumption works well for faults that do not depend on the robot's state, but other faults (e.g., collisions or getting lost) are very much localized in specific sets of states of the



(a)



(b)

Fig. 3: Time to fault detection as a function of the chosen fractional error ε. (a) shows all the experimental results, while (b) leaves out $\varepsilon = -0.05$ for ease of visualization of the rest of the data. Error bars mark one standard deviation of the data.

robot (e.g., a particular location, velocity or time). This may be part of the reason collision detection times were slower than in previous work. The authors are currently working on an extension of the monitor that will gather observations from particular areas of the robot's state space to find areas in which faults have occurred. Aside from such a model's obvious advantage of being able to detect localized faults more effectively, it will also contribute to the problem of fault isolation: the robot will be able to communicate not only that a fault has happened, but also in what region of the state space of the robot the fault happened (e.g., a fault happened in a certain area of the building, or only when the robot was going at a certain speed). Such a monitor would be significantly more helpful in the areas of fault isolation, and would allow for more useful fault communication to a human operator.

VI. Conclusion

This paper presented a probabilistic algorithm for fault detection during robot motion execution. The algorithm does not need a model of possible faults, but it instead looks for deviations from normal execution to detect failures. The algorithm thus does require a model of normal execution to be provided. For this paper, the model of normal execution was based on comparing equivalent information from redundant sources to find statistically significant differences between them, indicating a fault. Specifically, displacement data derived from wheel encoder output is compared to displacement data obtained from a particle filter-based localization algorithm to find statistically significant discrepancies that indicate a fault in execution.

Experimental results show that the algorithm can consistently detect faults and safely stop when the robot has a malfunctioning wheel encoder, with malfunctions as small as 5% off normal wheel encoder values. Further work will focus on two aspects: testing the monitor for detection of other safety-critical situations, such as collisions and getting lost, and expanding the algorithm to be able to detect and communicate localized faults, and not only global ones. Such further work will demonstrate the general applicability of the algorithm as a fault detection and isolation model.

References

[1] Joydeep Biswas, Brian Coltin, and Manuela Veloso. Corrective gradient renement for mobile robot localization. In *Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems*, pages 73 – 78. IEEE, September 2011.

[2] Joydeep Biswas and Manuela Veloso. Depth camera based indoor mobile robot localization and navigation. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation (to appear)*. IEEE, 2011.

[3] Brian Coltin, Manuela Veloso, and Rodrigo Ventura. Dynamic user task scheduling for mobile robots. In *Proceedings of the the AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots at AAAI 2011*, August 2011.

[4] Juan Pablo Mendoza, Manuela Veloso, and Reid Simmons. Motion interference detection in mobile robots. In *Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems (to appear)*, 2012.

[5] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1:652–659, 2004.

[6] Ola Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2):73–88, 2005.

[7] Mehdi Samadi, Thomas Kollar, , and Manuela Veloso. Using the web to interactively learn to find objects. In *Proceedings of the Twenty-Sixth AIII Conference on Artificial Intelligence*, 2012.

[8] Stephen Se, David Lowe, and Jim Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research*, 21:735–758, 2002.

[9] Manuela Veloso et al. Symbiotic-autonomous service robots for user-requested tasks in a multi-floor building. *Under submission*, 2012.

[10] Vandi Verma, Geoff Gordon, Reid Simmons, and Sebastian Thrun. Particle filters for rover fault diagnosis. In *IEEE Robotics & Automation Magazine special issue on human centered robotics and dependability*, 2004.

Risk-Variant Policy Switching to Exceed Reward Thresholds

Breelyn Kane and **Reid Simmons**
Robotics Institute
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
{breelynk,reids}@cs.cmu.edu

Abstract

This paper presents a decision-theoretic planning approach for probabilistic environments where the agent's goal is to win, which we model as maximizing the probability of being above a given reward threshold. In competitive domains, second is as good as last, and it is often desirable to take risks if one is in danger of losing, even if the risk does not pay off very often. Our algorithm maximizes the probability of being above a particular reward threshold by dynamically switching between a suite of policies, each of which encodes a different level of risk. This method does not explicitly encode time or reward into the state space, and decides when to switch between policies during each execution step. We compare a risk-neutral policy to switching among different risk-sensitive policies, and show that our approach improves the agent's probability of winning.

1 Introduction

Many probabilistic planners seek to maximize expected reward, and do little to incorporate the variance of the reward distribution when developing a plan for an agent. Therefore, many planners assume the agent has a risk-neutral attitude. In competitions, however, one often sees people behave differently (e.g., take more risks) when they believe they may end up losing. For instance, a sports team may play more aggressively when losing, but more defensively when trying to maintain a lead. This reflects the idea that it does not matter by how much one wins or loses, as long as the score is in the agent's favor. We model this as maximizing the probability of being above a given reward threshold (e.g. a competitor's current top score).

An agent needs to adjust its risk attitudes dynamically to exceed a threshold and win. For example, a campaign manager may appeal to particular advocacy groups or change the tone of the candidate's speech based on the candidate's position in the polls. In hockey, if a team is losing, they often remove the goalie in hopes that having an additional offensive player will increase the chances of a tying goal. This strategy is risky since it increases the chances of losing by more goals. If the hockey team were just trying to maximize its expected goal differential over the season it may never chose to remove the goalie. Intuitively, the idea of an agent needing to be more aggressive or conservative while executing a policy parallels a human's risk-seeking and risk-averse preferences.

A straightforward approach is to encode cumulative reward explicitly in the state space of a standard Markov Decision Process (MDP). Doing so, however, explodes the state space since the number of different cumulative reward values is typically very large. We propose an alternate approach in which multiple policies are generated offline. Then, an online algorithm decides which policy to use based on which is more likely to achieve the threshold constraint.

The algorithm reasons about the complete distribution of rewards, not just mean and variance, to make fine-grained decisions about which policy is most applicable for a given situation. In particular, we provide an algorithm that decides when to switch between strategies, at run-time, by estimating non-parametric distributions of the reward functions for each of these policies. Agents with risk-sensitive policies to choose from now have the ability to switch to a policy with a higher variance in hopes of increasing their chances of meeting the given threshold. We show that by switching policies in such a manner, the agent will end up doing better (with respect to the goal of finishing above a certain threshold of reward) than if it just followed the risk-neutral policy.

2 Background

2.1 Markov Decision Processes

Initially, our algorithm requires a model of the environment. We formulate this model as a Markov Decision Process, a mathematical representation of a sequential decision problem. An MDP is a four-tuple $\{S, A, T(S,A), r(S,A)\}$ consisting of:

- S : set of states $\{s_0 \ldots s_\infty\}$
- A : set of actions $\{a_0 \ldots a_\infty\}$
- $T(s_t, a_t)$: transition function yielding s_{t+1}, with $P(s_{t+1}|s_t, a_t)$
- $r(s_t, a_t)$: immediate reward function

Solving an MDP produces a policy, π, that maps states to actions $\pi : S \rightarrow A$. One approach is to use *value-iteration* to find a policy using the value-update rule below. This value function is also used to estimate the distribution of future discounted reward, as described in section 4.2.

$$V^{\pi}(s) = \max_{a \in A}\{R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')\} \quad (1)$$

2.2 Utility and Risk

Incorporating risk attitudes captures the trade-off between variance and mean in the distribution of reward outcomes. For instance, risk-seeking policies tend to have a lower mean and larger variance (more upside, but also more downside) than risk-neutral policies. Utility theory provides structure for making decisions of varying risk attitudes (Pratt 1964). A utility function maps an agent's value to a plan of wealth that represents the agent's rational choice. Linear utility functions maximize expected cumulative reward and represent risk-neutral decision makers, while exponential functions model risk-sensitive decisions. Concave utility functions reflect risk-averse decisions, and convex utility functions characterize risk-seeking decisions. The convexity of these functions changes for different risk factors.

When an agent's utility function is constant for the function duration (such as linear or exponential) the risk measure is constant, and this is known as a zero-switch utility function. Zero-switch utility functions are unrealistic, since decisions often change as wealth level changes. (Liu and Koenig 2008) take this a step further in defining MDPs that have a one-switch utility function. In the one-switch case, an agent acting risk-neutral may switch to being more conservative, which entails one switch from a linear function to a concave exponential function. While this is closer to realistic decision making, it seems more natural to allow the agent to switch multiple times between many utility functions, which is what our approach supports.

3 Related Work

While previous work has investigated switching between strategies (policies) to achieve different subgoals (Comanici and Precup 2010), our work instead considers adapting a strategy with the assertion of risk for a single goal – that of winning. In defining what it might mean to win, other works have discussed the idea of using thresholded reward objective functions (McMillen and Veloso 2007), (Wagman and Conitzer 2008). Our work differs by not requiring an adversary, by focusing on the use of risk attitudes, not requiring a threshold to be known ahead of time, and by having the ability to switch strategies during run-time. These works focus on solving variants of the MDP's objective function, and produce a single static policy. For instance, in (Geibel and Wysotzki 2005), risk is incorporated into the estimated objective function as a weighting factor.

By not focusing on altering the MDP's objective function, our work also trades off computation at execution time for creating policies more efficiently during planning time. This tradeoff was also a goal of work done by (Roth, Simmons, and Veloso 2005) in limiting communication for distributed agents modeled as DEC-POMDPs.

Another distinguishing characteristic of our work is that we reason about the complete distribution of a policy's reward, rather than just the expectation: in particular the reward is modeled as a non-parametric distribution. Other work that estimates the variance of an MDP (Tetreault, Bohus, and Litman 2007) does so by adding uncertainty to the MDP model's parameters. This is done by modeling the transition probabilities as a Dirichlet distribution and mapping confidence intervals over transition counts. Our approach better handles the tails of the distribution, which is very important for distinguishing the effects of different risk-sensitive policies.

4 Run-Time Policy Switching

A utility function that maximizes the probability of being over a given threshold, while representing the entire domain, is difficult to know ahead of time. This paper assumes that function is unknown, and emulates a multi-switch utility function by deciding which policy to follow (corresponding to different risk attitudes) during run-time. To accomplish this, we generate a suite of policies, including a risk-neutral policy (linear utility function), and risk sensitive policies (see section 4.1). A risk factor, δ, controls the amount of convexity for the exponential utility functions.

Then, for each policy, we estimate the complete reward distribution. This is done by executing (offline) a sufficient number of trajectories in the original MDP and collecting statistics on the cumulative rewards achieved at each state. The distribution of rewards is then modeled as a Cumulative Distribution Function (CDF) (see section 4.2). Finally, at each step during run-time, the agent determines which policy has the highest probability of exceeding the (user specified) reward threshold, given the current state and cumulative reward, so far. The agent then picks the action associated with the current state for that policy, executes it, then determines again which policy to use (see section 4.3).

4.1 Creating Policies

Creating a suite of policies that allow a variety of strategies for the agent to employ, requires a model of the world in the form of an MDP. While different techniques may be used to generate the various risk-sensitive policies, we use the transformation algorithm described in (Koenig and Simmons 1994) and (Liu 2005). This transformation does not affect the state space, but merely changes the structure of the MDP to choose actions based on probabilities that now form an exponential utility rather than a linear utility. The reason exponential utility functions are used is because they maintain the Markov property, preserve the decomposability of planning functions, and they are one of the most common risk functions. Convex utility functions (Figure 1a) are of the form:

$$U(r) = \delta^r, \delta > 1$$

and concave functions (Figure 1b) are of the form:

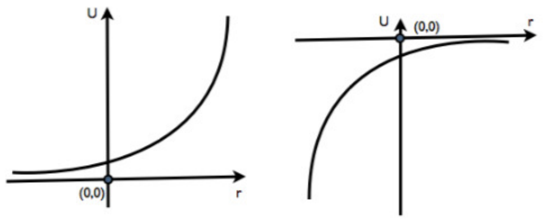
$$U(r) = -\delta^r, 0 < \delta < 1.$$

where δ is the risk factor, and r is a reward. (Liu 2005) further simplifies the function as :

$$U_{exp}(r) = \iota\delta^r,$$

where

$$\iota = sgn\ ln\ \delta.$$



(a) Convex (risk seeking) (b) Concave (risk averse)

Figure 1: Convex and concave exponential utility functions

To summarize the approach described in (Koenig and Simmons 1994), the transition probabilities of the original MDP are transformed by multiplying them by a function of risk and the immediate rewards are also transformed. Specifically, all non-goal state transition probabilities are converted to $P(s'|s,a)\delta^{r(s,a)}$. The original probabilities add up to one, but transforming the probabilities cause them to decrease depending on the risk factor. Since the transformed probabilities no longer add up to one, an additional sink state is introduced where the probability is $1 - \sum_{s' \in S} P(s'|s,a)\delta^{r(s,a)}$. The larger the risk factor, the greater chance the agent has of falling into this sink state, and the more it is encouraged to take riskier actions. Therefore, increasing the risk parameter, δ, generates policies that select increasingly riskier actions. Models with mixed positive and negative rewards, known as arbitrary rewards, require certain properties to hold, described in (Liu 2005). Positive rewards cause the initial MDP probabilities to be scaled to a value less than or equal to the reward, which might be greater than one, so arbitrary rewards are transformed to [0,1].

4.2 Estimating the Reward Distribution

The next step is to empirically estimate the (non-parametric) distribution of reward for every state of each policy. We do this by executing the policies in the original MDP. Each trajectory run will result in a cumulative discounted reward value. These values make up the distribution.

The cumulative discounted reward is given by:

$$R(s) = \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \tag{2}$$

where γ is the discount factor.

More explicitly:
$a = \pi(s)$
$R(s) = \gamma^0 r(s_0, a_0) + \gamma \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)$
$R(s) = r(s_0, a_0) + \gamma(r(s_1, a_1) + \gamma(r(s_2, a_2) + \ldots))$
where T(s,a) is the transition function for generating all of the next states:

$$R(s) = r(s,a) + \gamma(R(T(s,a))) \tag{3}$$

Note that the discount factor decreases the contribution of the future reward term over time. Therefore, there is a point where the discount factor causes the future reward to be arbitrarily small $\gamma^{Tr} * maxR \leq \epsilon_1$, where $maxR$ is the maximum possible immediate reward and ϵ_1 is a small constant. Trajectory length (Tr), or required number of time-steps, is then calculated as:

$$Tr = \frac{\log(\epsilon_1) - \log(maxR)}{\log(\gamma)} \tag{4}$$

A trajectory is a sequence of state and action transitions $s_0 \rightarrow^{a_1} s_1 \rightarrow^{a_2} s_2$ generated by following the known policy in the environment, and may visit the same state multiple times. The stopping criteria, for how many trajectories to run, is based on the convergence of a fourth-order statistic. The statistic needs to be scale invariant, since our approach is domain-independent. Convergence occurs when the variance of the distribution variance divided by the mean of the variance is less than some small value, ϵ_2. This statistic states that convergence occurs when the spread between numbers in sample distributions (obtained from the overall distribution) is arbitrarily small, and then this is scaled by the mean.

After collecting the distribution of values for each policy, we convert them into the corresponding Cumulative Distribution Function (CDF).

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t = P(V \leq x) \tag{5}$$

The CDF, $F(x)$, gives the probability of achieving reward less-than-or-equal to some cumulative discounted reward, x. Note that for each policy we need a separate CDF for every state. Example CDFs are shown in Figure 4 and pseudocode for estimating the reward function is presented in Algorithm 1. For every state, the trajectory length is calculated by equation (4). Then, while the statistic has yet to converge, lines 7-10 go through an entire trajectory sequence saving off the immediate reward that corresponds with each state. The next *for* loop is used to discount all the states' corresponding reward values at once.

To increase efficiency, we simultaneously collect rewards for every state visited along a trajectory. In particular, the cumulative value of s_t is $R(s_t)$ as given in equation (2). Trajectories can visit the same state multiple times, so one trajectory run may collect multiple values for that state. We run trajectories of length $2 * Tr$ (line 6 of Algorithm 1), but do not include values of states s_t where $t > Tr$. Even with having to run each trajectory twice as long, collecting multiple values per trajectory is a huge win in our experiments, at least an order of magnitude faster.

4.3 The Switching Criteria

It is straightforward to calculate the maximum probability over a threshold using the CDF. The probability of being above a discrete threshold is a matter of subtracting the CDF from one.

$$1 - F(x) = \int_{x}^{\infty} f(t)\,\mathrm{d}t = P(V > x) \tag{6}$$

Algorithm 1 generates reward distributions given a policy. For each starting state, a trajectory is run for the calculated trajectory size. The second nested for loop is used to discount the rewards by backtracking.

```
1:  for $i = 1 \rightarrow stateSz$ do
2:      $Tr = getTrajectorySize(\epsilon_1)$
3:      do
4:      $vals = getAllValsSavedForStartState(s_i)$
5:      $statistic = var(var(vals))/mean(var)$
6:      for $j = 1 \rightarrow 2 * Tr$ do
7:          $a_j = \pi(s_j)$
8:          $r = r(s_j, a_j)$
9:          $saveOff(r, s_j, 0, j)$
10:         $s_{j+1} = getNextState(T(s_j, a_j))$
11:     end for
12:
13:     //backtrack to discount the values
14:     $R = 0$
15:     for $x = savedOffValsSz \rightarrow 1$ do
16:         $r = getSavedOff(x).ImmedRew$
17:         $R = r + \gamma * R$
18:         if $x < Tr$ then
19:             $s = getSavedOff(x).s$
20:             $saveOff(r, s, R, x)$
21:         end if
22:     end for
23:     while($statistic > \epsilon$)
24: end for
```

We define $R^t(s)$ as the *running* cumulative discounted reward for time t, starting in state s.

$$R^t(s) = \sum_{i=0}^{t} \gamma^i r(s,a) \tag{7}$$

[Note that $R(s)$ in equation [2] is then equal to $R^\infty(s)$.]

Now, $P(V > x)$, becomes:

$$P(R(s_0) > thresh|\pi)$$

$$R(s_0) = \sum_{i=0}^{t-1} \gamma^i r(s,a) + \sum_{i=t}^{\infty} \gamma^t \gamma^{i-t} r(s_i, a_i)$$

$$R(s_0) = R^{t-1}(s_0) + \gamma^t R(s_t)$$

so to maximize the probability of being greater than threshold $thresh$, we have:

$$\max_\pi P(R^{t-1}(s_0) + \gamma^t R(s_t) > thresh|\pi)$$

$$\max_\pi P(R(s_t) > \frac{thresh - R^{t-1}(s_0)}{\gamma^t}|\pi)$$

$$valtofind = \frac{thresh - R^{t-1}(s_0)}{\gamma^t}$$

The pseudocode in Algorithm 2 details the process of selecting actions, by choosing the policy that maximizes the probability of being above *valtofind*. Note that, if the policies are equal, the algorithm defaults to following the risk-neutral policy. Also note that the CDF for each policy is not required to be smooth, and in some cases may resemble a step function. It is necessary to interpolate the values to retrieve the probability since the CDF is not continuous. This occurs on lines 7 and 8 (of Algorithm 2) in the $cdfGet$ function.

Algorithm 2 executes actions starting in some initial state. It switches between selecting the actions from a set of policies based on a threshold.

```
1:  Given threshold $thresh$, start state $s_0$
2:  $Tr = getTrajectorySize(\epsilon)$
3:  $r_0 = 0$
4:  for $i = 0 \rightarrow Tr - 1$ do
5:      $r_i = r(s_i, a_i)$
6:      $cdfVal = (thresh - R_i)/\gamma^i$
7:      $max = 0$
8:      $\pi_{currBest} = \pi_{risk-neutral}$
9:      for $0 \rightarrow allcdfs$ do
10:         $cCurr = 1 - cdfGet(cdfVal, cdf.this)$
11:         if $cCurr > max$ then
12:             $max = cCurr$
13:             $\pi_{currBest} = \pi_{currCDF}$
14:         end if
15:     end for
16:     $a_{i+1} = \pi_{currBest}(s_i)$
17:     $R_{i+1} = R_i + \gamma^i * r_i$
18:     $s_{i+1} = getNextState(T(s_i, a_{i+1}))$
19: end for
```

4.4 Changing the Reward Threshold

Our formulation assumes that the reward threshold is given as an input. In some cases, threshold values can correlate to interpretations of the world, such as cut-off times for delivering items or the current high score of a video game. A threshold could also be a percentage line in the CDF, such that 60% of the time the distribution is better than some value. The threshold depends on the problem one is trying to solve. The algorithm does not care how the threshold is chosen or what it represents. The algorithm attempts to maximize the probability of being over the threshold, regardless.

Note, however, that as the threshold shifts to the lower end of the reward distribution, the agent chooses policies that are more risk-averse; as the threshold shifts to the other extreme, the agent chooses more aggressive policies. Depending on the uncertainty in the environment, it may need to switch to risky policies earlier, rather than later, or scale back to a less risky policy when it is performing well.

5 Evaluation

We have tested our algorithm in two domains: *Super Mario Bros* (described in section 5.1) and a simpler pizza delivery domain (described in section 5.4).

5.1 Mario Domain

The *Super Mario Bros* domain uses the Infinite Mario simulator [1]. Previous work using this domain (Mohan and Laird 2011) compared a learning agent to the agent provided with the simulator for one world. Our agent generalizes a small state space over many worlds.

Infinite Mario includes a trainer for generating episodes of varying difficulty and type. The agent (Mario) must move through the environment collecting objects, fighting monsters, all while getting to a goal line without being killed. The environment is made up of observations and actions. Mario receives the visual scene as a 16 x 22 array of tiles. Each tile contains a character or a number. If the tile is a character, it represents the tile type (coin, block, etc); if it is an integer, it indicates how Mario can travel through that tile. There is also an observation vector that includes the location and types of monsters in the environment. The primitive actions Mario can take correspond to buttons on a Nintendo game controller: direction, jump, and speed.

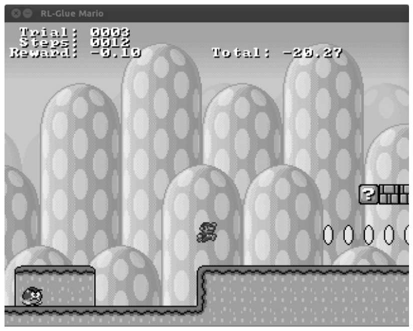


Figure 2: Mario world (screen capture from the Infinite Mario simulator)

The action space for our model is made up of nine macro-actions (see Table 1), inspired by Function Level Operators in (Mohan and Laird 2010). A macro-action tries to find a path to its object of interest using A*. The A* path goes around anything Mario is not able to travel through on his way to the object. The macro-action executes the A* path until the end condition is met, using the appropriate combination of primitive actions.

The state space for our MDP is based on Mario's relationship to objects in the environment. A symbolic state representation for Mario was presented in (Mohan and Laird 2010). Our state vector contains seven dimensions {mario, block, coin, mushroom, pipe, finish, goomba}. Each dimension takes on two possible values; either the object is near (in the visual scene) or far (not in the visual scene). The "mario" dimension indicates if Mario is big or small.

In Infinite Mario, each episode is generated based on a seed value. The rewards Mario receives are -0.01 for each time step, -10 for dying, 1 for collecting a coin or killing a monster, and 100 for reaching the finish line. We ran 1,000 trials over different starting episodes to estimate the transition probabilities and cumulative rewards for macro-actions.

[1] http://2009.rl-competition.org/mario.php

Macro-actions		
Action Name	Description	End Condition
grabCoin	go to nearest coin	past coin
grabCoinForever	go to nearest coin	no coins
avoidMonster	go past monster	past monster
tackleMonster	go to above monster (to smash)	past monster
tackleMonster Forever	go to above monster (to smash)	all monsters smashed
searchBlock	hit nearest question block	past block
searchBlock Forever	hit nearest question block	blocks searched
getMushroom	find hidden mushroom	past mushroom
moveSmart	use A-Star to move right 4	past move position

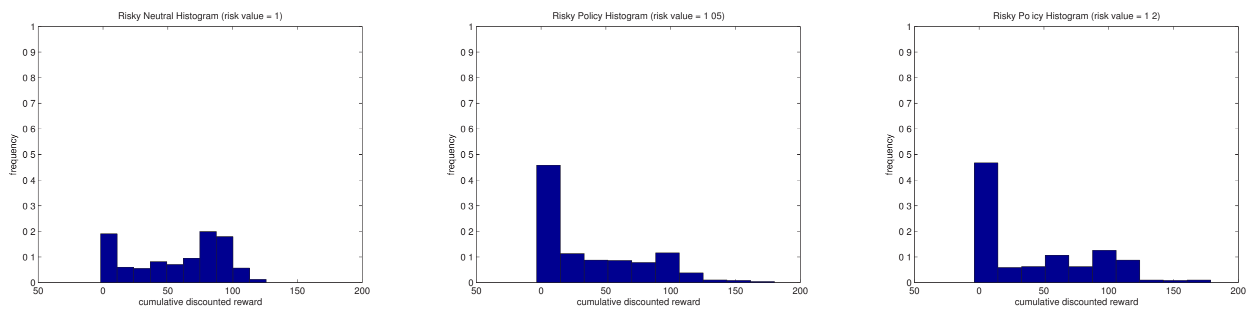
Table 1: Macro-actions of the MDP model

Mario chooses randomly from the set of macro-actions at each time step (biased to moving towards the finish line, in order to avoid getting stuck too often). While the macro-action is executing, immediate rewards are accumulated, and these become the "immediate" reward of the macro-action. Similarly, the state when the macro-action is started and the state when it completes are used to update the transition probability for that pair of states.

The MDP is solved using value-iteration, where the immediate reward and transition functions return values based on the information captured by sampling the Mario world. To generate each risky policy, the probability changes to $P(s'|s,a)\delta^{r(s,a)}$, where $P(s'|s,a)$ is the transition probability obtained from earlier testing, δ is a risk factor greater than one, and $r(s,a)$ is the average immediate reward for this state and action from the recorded rewards. The Mario world contains both negative and positive rewards, so the probability returned must be linearly transformed to a range between 0 and 1. In other words, the interval $[0, 1 * \delta^{maxReward}]$ needs to map to $[0,1]$. Exponential functions maintain their convexity for affine transforms. The linear transformation for the probability mapping is just a mapping from $[A,B]$ to $[C,D]$ where $x' = ((D-C)/(B-A))x + C$.

The policies produced for the risk-seeking transformation tend to choose the *forever* macro-actions more (see Table 1). For these actions, there is a chance of getting a higher score by retrieving all the objects and an increased chance of dying, since Mario is in the environment longer. The *moveSmart* macro-action also is chosen more often in the risky policies. This action may be chosen more because it produces more variance in the environment than just going after a particular object. The histograms for various policies are displayed in Figure 3. Not that, as the risk value increases, the distributions have a larger variance but lower mean.

The reward functions for each state in the MDP are estimated according to the algorithm described in section 4.2.



(a) risk-neutral policy, risk value 1 (b) risk value 1.05 (c) risk value 1.2

Figure 3: Histograms for varying risk values

Two variations of the domain were used. The Mario world modeled as an MDP and the actual Infinite Mario simulator. The difference between the two can be thought of as an example where the model is well known and one where things may not be modeled perfectly.

Reward distributions are estimated for each state in these domains. For the Infinite Mario simulator, estimations are taken over multiple random seeds (different episodes) and, for the MDP model, domain trajectories are sampled based on the transition probabilities constructed previously. The number of samples needed depend on the convergence statistic described in section 4.2. CDFs are then constructed for all of these states. The CDF for various policies in start state 0 {mario small, block far, coin far, mushroom far, pipe far, finish far, goomba far} is shown for the MDP domain in Figure 4a and the actual Mario simulator in Figure 4b. The MDP modeled Mario is slightly more optimistic, partly because the model was constructed using the average immediate rewards collected in the environment. Also, it is possible the MDP assumes that there are more transitions to states with higher reward (such as more coins) than actually exist in the real environment.

5.2 Mario Results

Different threshold values affect how often the switching strategy chooses more risky actions. The rewards are discounted so the total reward received is lower than the exact values returned in the Mario simulator. First, the results are displayed for exploiting the policy in the modeled Mario MDP (Tables 2 and 3). Each group of results compares 1,000 trajectory runs, all with start state 0, using just the risk-neutral policy versus 1,000 runs using the switching strategy (switching between the risk-neutral policy and a risky policy with $\delta = 1.2$). In order to compare trajectories fairly, random probabilities are generated offline on a per-state basis. As the policy is being exploited in the original MDP, the probability that corresponds to the current state, and number of times visited, is retrieved. This probability is then the same for both trajectories, and is used to determine the next state.

Table 4 shows results in the actual Mario world for the risk-neutral policy and the switching strategy (switching between the risk-neutral policy and a risky policy with $\delta =$

Mario MDP model for a threshold of 30:		
	Switching Wins	Switching Loses
Risk-neutral Wins	754	18
Risk-Neutral Loses	**59**	169
No switching lost : 228 times		
Switching strategy lost: 187 times.		

Table 2: Fails 4.1% less using switching strategy; reduces losses by 18%.

Mario MDP model for a threshold of 100:		
	Switching Wins	Switching Loses
Risk-neutral Wins	40	41
Risk-Neutral Loses	**222**	697
No switching lost : 919 times		
Switching strategy lost: 738 times.		

Table 3: Fails 18.1% less using switching strategy; reduces losses by 19.7%

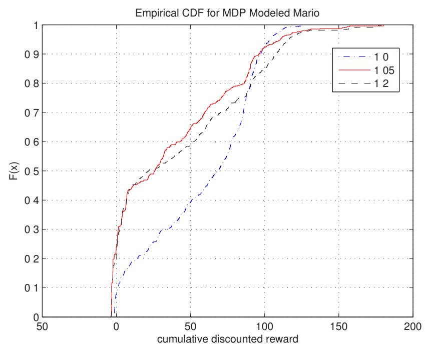
Mario simulator for a threshold of 30:		
	Switching Wins	Switching Loses
Risk-neutral Wins	72	90
Risk-Neutral Loses	**107**	731
No switching lost : 838 times		
Switching strategy lost: 821 times.		

Table 4: Fails 1.7% less using switching strategy; reduces losses by 2%

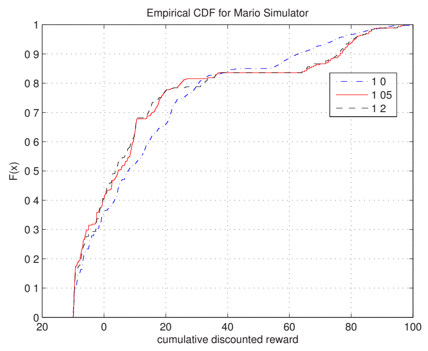
1.05). Each group of results compares runs over 1,000 different worlds, generated using different seeds.

5.3 Mario Discussion

The results for trajectories run in the MDP model demonstrate how beneficial the switching strategy is when the model is well known. As the threshold value increases the



(a) State 0, Mario MDP model CDF



(b) State 0, Mario simulator model CDF (collected for the same start state)

Figure 4: CDFs for the Mario MDP model and the Infinite Mario simulator for start state 0. The graphs show the estimated reward function of the risk-neutral policy (δ=1) compared to risky policies ($\delta = 1.05, \delta = 1.2$) for each domain.

switching strategy is more beneficial because the higher threshold takes riskier actions sooner, which allows the resulting CDF to reside in between the risk-neutral and risky policy at higher values. Low thresholds may never take risky actions soon enough to reach the higher cumulative discounted reward values.

The results for trajectories in the Mario simulator show that even in a world that may not be modeled perfectly the switching strategy can provide some benefit. There is no improvement when using a threshold of 80, but in Figure 4b one can see that a threshold this high is approaching the upper bound for what the agent can achieve in practice.

5.4 Pizza Domain

We also evaluated our algorithm in a navigation domain, where a vehicle drives through a non-deterministic world for the purpose of delivering a pizza. The idea is that the delivery driver may need to be risk-sensitive in order to make the delivery on time. The state space has three dimensions: an x,y location and a Boolean value indicating whether the driver has a pizza. For a ten by ten grid, the world contains 200 states (refer to Figure 5).

The world contains the following actions: (PICKUP, DROPOFF, MOVE{N,S,E,W}, RISKMOVE{N,S,E,W}). The MOVE{N,S,E,W} actions are more deterministic and have a higher cost. Riskier actions (RISKMOVE{N,S,E,W}) are cheaper and have a chance of traveling further, but have a lower probability of actually progressing. For action MOVE{N,S,E,W}, there is an 80% chance of moving one square and a 20% of staying in the same square. The probabilities for the action RISKMOVE{N,S,E,W} are a 9% chance of moving one square, a 7% chance of moving two squares, and an 84% of staying put. Table 5 presents the immediate rewards for this domain.

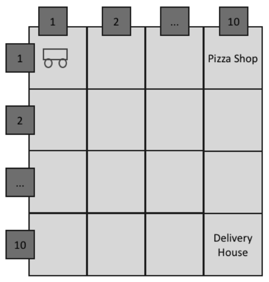


Figure 5: Navigation grid for pizza delivery world

Reward Mappings		
Action Name	State	Reward
DROPOFF	x,y=delivery location, have pizza	50
PICKUP	x,y=pizza shop, no pizza	-1
RISKMOVE{N,S,E,W}	any	-2
MOVE{N,S,E,W}	any	-6

Table 5: Reward mappings of the MDP model

5.5 Pizza Domain Results

The results in Tables 6 and 7 compare the risk-neutral policy and a risky policy with risk factor $\delta = 1.2$, at two different threshold values. The policies generated are run in the original MDP domain. Each group of results compares 10,000 trajectory runs between the risk-neutral policy versus the switching strategy. As before, we generate random probabilities offline on a per-state basis to compare the policies fairly.

As stated in the introduction, a straightforward approach to the problem of exceeding reward thresholds is to encode cumulative reward explicitly in the state space. Since the pizza delivery domain is small enough, we can feasibly do that and compare the results against our approach. The state space of the pizza domain was augmented to include an additional dimension of cumulative reward. We capped the maximum and minimum cumulative reward from 0 to -150 and removed discounting. This inflates the 200 states

For a 30% threshold (threshold = -100):		
	Switching Wins	Switching Loses
Risk-neutral Wins	6422	458
Risk-Neutral Loses	**1412**	1708
No switching lost : 3120 times		
Switching strategy lost: 2166 times.		

Table 6: Fails 9.5% less using switching strategy; reduces losses by 30.6%

For a 60% threshold (threshold = -88.5):		
	Switching Wins	Switching Loses
Risk-neutral Wins	2885	830
Risk-Neutral Loses	**3271**	3014
No switching lost : 6285 times		
Switching strategy lost: 3844 times.		

Table 7: Fails 24.4% less using switching strategy; reduces losses by 38.8%

to 30,000 states. The MDP reward function now returns a large positive reward if the driver delivers a pizza at the goal and the cumulative reward exceeds the reward threshold and a small positive reward for delivering the pizza while not exceeding the threshold, to encourage the planner to achieve the goal regardless. An additional cost is also added for states that are not the goal and are under the threshold value. This parallels the MDP transformation necessary to use thresholded reward objective functions as explored in (McMillen and Veloso 2007).

As expected, the offline planning times for the reward-augmented policy were significantly greater than for our algorithm. Solving for the policy with 30,000 states took approximately 18 hours, while solving for the 200 state policy took a few seconds (using an Intel 3.15 Ghz processor). Even though our algorithm must solve multiple policies, generate offline reward distributions for each state (which took 5 to 10 minutes per policy) and construct the corresponding CDFs (which took about 1 minute per policy), the processing is still significantly less than it takes to solve for the augmented-reward policy.

The difference in run time computation is small. Our approach must evaluate a point on the CDF for each policy, which is a straightforward linear interpolation. There is a cost for reading in the CDFs (which are generated offline) for each state of each policy, but that is done just once, at start up.

Comparing the results of the policies did not show a significant difference between the reward-augmented and switching policies. Both performed better than the risk-neutral policy, but their similarity could be attributed to the fact that in such a simple domain the switching strategy is approaching optimal. In general, the reward-augmented policy is expected to perform better (and should behave optimally, with respect to the objective of exceeding the reward threshold).

5.6 Pizza Domain Discussion

Besides performing significantly better than the risk-neutral policy, it is interesting to note that the average trajectory lengths are higher for the switching strategy versus the following the risk-neutral policy. Also, the trajectory lengths for the switching strategy increase as the threshold increases because as more risky actions are taken there is a higher chance of getting stuck in the same state.

Policies generated contained all MOVE actions for the risk-neutral case and more RISKMOVE actions depending on the convexity of the risk factor. The reward-augmented policy chose more risky actions as the cumulative reward got closer to the threshold, and returned to MOVE actions once the threshold was exceeded.

While increasing the state space to include cumulative reward creates an optimal policy for thresholded rewards, there is a tradeoff with longer execution time and less flexibility for setting the threshold. The optimal policy must be re-generated for every threshold that needs to be tested. This can take days depending on the size of the state space. Our algorithm allows a threshold to be re-set during the switching stage, and does not affect the offline policy generation.

6 Conclusion

For these specific domains, there was not a large difference between the risk-seeking levels, so results are shown only comparing risk-neutral with one risky policy. This algorithm allows for more complex domains to compare with multiple risk-sensitive policies based on the architect's preferences.

We presented a domain-independent algorithm that aims to maximize the probability of exceeding a threshold at execution time using risk-sensitive policies. This was demonstrated on two domains showing the benefits of taking more risks to win. For future work, we would like to continue to explore additional ways an agent adapts and operates reliably in a dynamic environment. More specifically, having the agent gather more contextual awareness on whether it was winning or losing is useful. The long term goal is to apply these principles to enhance the robustness of real robotic systems.

7 Acknowledgments.

This work is supported by AFOSR grant #FA2386-10-1-4138. We thank Sven Koenig and Yaxin Liu for their help with the risky policy transformations, and thank John Laird and Shiwali Mohan for their assistance with the Super Mario domain and simulator.

References

Comanici, G., and Precup, D. 2010. Optimal policy switching algorithms for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 709–714. International Foundation for Autonomous Agents and Multiagent Systems.

Geibel, P., and Wysotzki, F. 2005. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research* 24(1):81–108.

Koenig, S., and Simmons, R. 1994. How to make reactive planners risk-sensitive. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, volume 293298.

Liu, Y., and Koenig, S. 2008. An exact algorithm for solving mdps under risk-sensitive planning objectives with one-switch utility functions. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, 453–460. International Foundation for Autonomous Agents and Multiagent Systems.

Liu, Y. 2005. *Decision-theoretic planning under risk-sensitive planning objectives*. Ph.D. Dissertation, Citeseer.

McMillen, C., and Veloso, M. 2007. Thresholded rewards: Acting optimally in timed, zero-sum games. In *Proceedings of the national conference on artificial intelligence*, volume 22, 1250. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Mohan, S., and Laird, J. 2010. Relational reinforcement learning in infinite mario. *Ann Arbor* 1001:48109Y2121.

Mohan, S., and Laird, J. 2011. An object-oriented approach to reinforcement learning in an action game. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Pratt, J. 1964. Risk aversion in the small and in the large. *Econometrica: Journal of the Econometric Society* 122–136.

Roth, M.; Simmons, R.; and Veloso, M. 2005. Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 786–793. ACM.

Tetreault, J.; Bohus, D.; and Litman, D. 2007. Estimating the reliability of mdp policies: a confidence interval approach. In *Proceedings of NAACL HLT*, 276–283.

Wagman, L., and Conitzer, V. 2008. Strategic betting for competitive agents. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, 847–854. International Foundation for Autonomous Agents and Multiagent Systems.

2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Dimensionality Reduction for Trajectory Learning from Demonstration

Nik A. Melchior and Reid Simmons

Abstract— **Programming by demonstration is an attractive model for allowing both experts and non-experts to command robots' actions. In this work, we contribute an approach for learning precise reaching trajectories for robotic manipulators. We use dimensionality reduction to smooth the example trajectories and transform their representation to a space more amenable to planning. Key to this approach is the careful selection of neighboring points within and between trajectories. This algorithm is capable of creating efficient, collision-free plans even under typical real-world training conditions such as incomplete sensor coverage and lack of an environment model, without imposing additional requirements upon the user such as constraining the types of example trajectories provided. Experimental results are presented to validate this approach.**

I. Introduction

Precise reaching and manipulation are important skills for robots that operate in real-world environments. Assembly-line robots align pieces of hardware, attach bolts, and weld seams. Humanoids need to precisely grasp and place objects. The motions performed by these robots must often be tediously scripted by a programmer or technician familiar with the capabilities and limitations of the particular robot in use.

This work seeks to make it easier for both experienced and novice robot users to command precise motions by providing examples. The motions are demonstrated by moving a robotic arm in passive mode or through a teleoperation interface. This kinesthetic form of training is intuitive for users, and it avoids the problem of mismatched models when learning from methods such as human motion capture.

Demonstration is also attractive because it allows the human to convey implicitly the location of obstacles (by avoiding them) and non-geometric constraints. Since these robots typically operate in sensor-poor environments, it would be difficult to place a ladar or stereo camera pair in a position capable of observing the entire workspace of a high degree of freedom (DOF) dexterous arm, particularly when accounting for occlusion by the arm itself. Building a precise model of the workspace by hand is a time-consuming and error-prone task. However, knowledge of free space can be inferred from the space occupied by the robot during kinesthetic demonstrations.

Non-geometric constraints are more subtle, but are also conveyed by the demonstrated trajectories. For example, the task may require the robot's end effector to remain in a certain orientation throughout execution, or to avoid a portion of the workspace shared with another robot or human, even if that area is not currently occupied. Both of these types of constraints are conveyed implicitly by the human teacher through demonstrations, and may be incorporated into learned trajectories.

Robotics Institute of Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, USA {nmelchio, reids}@cs.cmu.edu

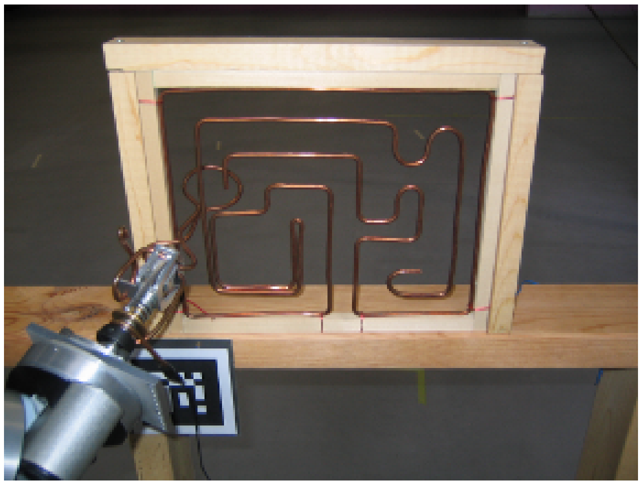


Fig. 1. The experimental platform: a Barrett WAM 7-DOF arm navigating a wire maze.

In this work, we present a method for learning robot manipulator trajectories from expert or novice demonstrations. We contribute a neighbor-selection technique for finding similar sections of demonstrated trajectories without the parameter selection required for previous methods. Using dimensionality reduction, we create a task-specific planning space in which collision-free trajectories may be created, even without a model of the environment in which the robot operates. Figure 1 shows the wire maze used in one set of experiments. Participants were asked to manually move the robot manipulator to navigate the wire maze, producing trajectories such as those in figure 5(a). The robot learner used these examples to produce novel trajectories capable of navigating the maze from a variety of initial conditions.

II. Related Work

The problem of learning trajectories from examples occupies an interesting niche between the traditional fields of motion planning and machine learning. Motion planning algorithms typically choose actions for the robot to execute based on knowledge about the environment gathered by sensors or by some other means. Grid-based planners such as A* and D* [1] or randomized sampling-based planners such as RRT [2] and probabilistic roadmaps [3] all depend on knowledge of free space that the robot is allowed to occupy. Whether they use a binary occupancy grid or a graduated costmap, traditional motion planning techniques will not operate directly on the examples provided. Without an explicit model of the environment in which the robot operates, obstacles may be conservatively inferred to exist in any region of the workspace not visited by the robot during training.

Traditional machine learning techniques tend to be more appropriate to this domain, but learning can be difficult to generalize properly. A single trajectory of a 7-DOF robotic arm may contain hundreds of points in a configuration space including position, velocity, and perhaps a few other features of individual points. For example, velocities may need to be included in the configuration space if dynamic constraints are to be respected. Providing examples of every area of this configuration space would be too time-consuming, so the learning algorithm must generalize examples correctly. Without knowledge of obstacles, though, simple approaches such as k-nearest neighbor can easily generalize too broadly, and other techniques must be used to correct this [4]. Other work in Reinforcement Learning [5] seeks to correctly generalize training examples as much as possible. Rather than drawing generalizations from individual points, other strategies use short portions of trajectories as their learning primitives. Several approaches use an intermediate representation such as domain-specific primitives [6], basis functions [7], or various types of splines [8], [9], [10] to improve the fit of example trajectories. Recent work using Gaussian Mixture Models has focused on limiting the number of necessary training examples while ensuring confident execution of learned behaviors [11], or ensuring correct behaviors even when online adaptation is required [12]. Another recent work [13] attempts to learn the cost functions implicitly used by the human teacher in generating entire examples.

Unfortunately, these approaches typically lack the greatest strength of the previous category of algorithms: since obstacles are not modelled, no guarantees can be made as to the safety of a planned path. One method for dealing with this problem is to present the planned path to a user in a graphical interface, providing an opportunity to correct or reject the planned path [14], [10]. Unfortunately, this introduces the requirement that the environment be precisely modelled. Another promising approach, provided that collisions are not catastrophic or costly, allows the user to mark portions of generated or example trajectories as undesirable [15] after they are executed. Delson and West [16] introduced an algorithm that, with certain assumptions, including a limit of two dimensions, ensured that learned trajectories would be collision-free. However, their work did not account for redundant manipulators, and imposed the constraint that all example trajectories must be homotopically equivalent. That is, all examples must take the same route between obstacles. While this is not an arduous restriction in simple cases, it may be difficult to ensure homotopic equivalence in environments with more obstacles or in higher dimensional spaces.

Our approach to safety requires a model of the robot, but not the environment. This requirement should be easy to fulfill since robots change far less often than the environments in which they operate, and many identical robots are generally produced with the same hardware configuration. Given the example trajectories, it is straightforward (though computationally expensive) to determine all areas of the workspace that have been occupied by the robot. Next, every grid cell in a discretized representation of the configuration space may be marked as safe if the robot, in that configuration, occupies only areas of the workspace that were occupied during training. The user may also elect to provide a conservative buffer around the robot's position that is also considered safe to occupy.

III. Approach

In this work, we seek to develop an approach that combines the strengths of previous programming by demonstration and planning systems. This system will learn to perform a precise reaching task with a dexterous arm from a variety of initial conditions. The generated trajectories should be guaranteed collision-free in static environments despite limited sensing and no explicit model of the environment. Finally, the system should be intuitive and easy to use, even without training or extensive knowledge of the algorithm used.

Our approach uses dimensionality reduction to transform the example trajectories to an intrinsic embedding suitable for learning the particular task at hand. This retains explicit representation of each of the examples provided while smoothing some noise and jitter, and providing a more convenient domain in which to plan. By combining the example trajectories with a model of the manipulator, the free area in the workspace may be determined. A conservative planner should assume that any space not occupied by the manipulator during training may contain an obstacle. Once the intrinsic task embedding is discovered, a novel plan may be created in the low-dimensional space and transferred, or *lifted*, to the original control space.

We will first examine the strategy and motivation for planning in a reduced-dimensionality space. Next, we present our extension to Isomap [17] for time-series data. Finally, we present an implementation of the complete system on a 7-DOF robotic arm and experimental results are discussed.

A. Dimensionality Reduction

Our primary motivation for dimensionality reduction is to ease the task of planning by discovering an intrinsic embedding for examples of the task. That is, rather than planning arbitrary trajectories through the robot's workspace or configuration space, we create a *task space* in which the desired trajectory is as simple as possible. In our work to date, a two-dimensional task space has been used to represent full 6-DOF trajectories using a redundant 7-DOF manipulator. One dimension represents time, or progress through the task, while the other dimension represents variations between distinct examples.

However, finding an intrinsic low-dimensional representation of the data has other benefits. Dimensionality reduction is typically used in Programming by Demonstration to deal with the correspondence problem [18] between high-dimensional training data (such as human motion capture) and low-dimensional controls (such as dexterous arm joint angles). When kinesthetic demonstrations are used, the training data is collected in the robot's control space. However, dimensionality reduction is still useful as it accentuates commonality among multiple training examples, and eliminates much of the noise (due to imperfect sensors) and jitter (due to imperfect human motion) that needlessly distinguishes them. Smoothing is achieved because the lower dimensionality space lacks the degrees of freedom to precisely represent every aspect of the example trajectories. Thus, the high frequency noise is eliminated while the features common to all examples are emphasized. This strategy is to be favored over other techniques that smooth trajectories one at a time by removing high-frequency or low magnitude variations. Although neither approach requires domain knowledge for its application, dimensionality reduction is able to preserve features common to many trajectories, even at the same magnitude as the noise, because it operates on all the trajectories at once. For instance, dimensionality reduction smooths out random jitters, but can maintain small "bumps" that are common across trajectories.

One promising technique for dimensionality reduction is Isomap. This algorithm finds a non-linear embedding for data using geodesic distances between nearby points. In essence, it discovers a lower-dimensional manifold embedded in the original space. The input data points lie on (or near) this manifold, so they can be represented in fewer dimensions. Isomap operates by first constructing a neighborhood graph connecting all of the points. A matrix of all-pairs shortest path distances are computed over this neighborhood graph. Finally, the low-dimensional embedding is constructed by applying Multi-Dimensional Scaling (MDS) to the distance matrix. This creates a space in which geodesic (graph-based) distances are preserved.

The original Isomap algorithm was not specificly tailored for time-series data, but it may be applied to points sampled from trajectories. One extension to Isomap, Spatio-Temporal Isomap [19], attempts to exploit the inherent relationships between these points to improve the embedding. ST-Isomap introduces changes to the first two steps of Isomap: construction of the neighborhood graph and computation of the distance matrix. First, it exploits the obvious neighbor relationships inherent in time-series data. Adjacent samples from a single trajectory, which we call *temporal neighbors*, are clearly related, and are thus linked in the neighbor graph. ST-Isomap also attempts to discover what we call *spatio-temporal neighbors*, or neighbors in different trajectories that occur at the same time in the task. For each of these types of neighbors, a tunable parameter is used to reduce the perceived distance between linked points. Although ST-Isomap is able to produce reasonable embeddings for many tasks, we have found that the tunable parameters must be chosen accurately for the nature and scale of the task at hand. In this work, we attempt to refine the neighbor selection mechanism in order to obviate the need for parameter selection and produce a better embedding.

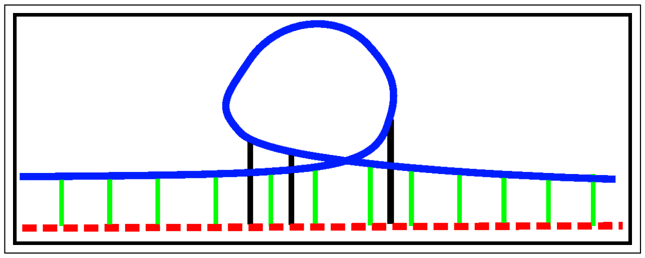


Fig. 2. Regularly spaced neighbor links from the solid (blue) trajectory to the dashed (red) trajectory result in some undesirable links (black).

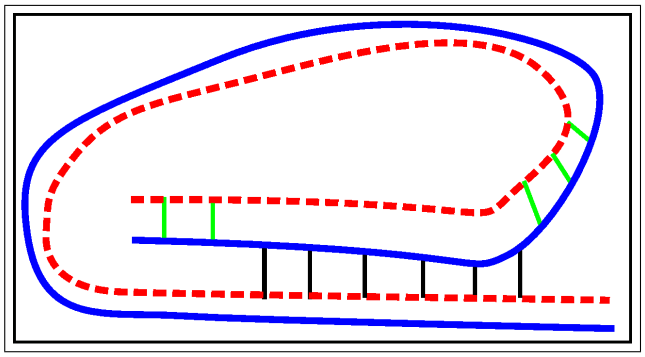


Fig. 3. The solid (blue) trajectory drifts closer to distant sections of the dashed (red) trajectory. Pointwise nearest-neighbor alone is not sufficient to correct this problem.

B. Neighbor Selection

Neighbor selection is the key to discovering an intrinsic task embedding for time-series data. Although humans can typically discern global structure even in noisy collections of points, it is a challenging task for a robot learner. This problem, known as graph or manifold denoising, has been studied extensively for application to Isomap [20] and other graph-based learning algorithms[21], [22], [23]. Time-series data presents specific challenges to this effort, but it also has the advantage that part of the structure of the data is known. Specifically, we retain the temporal links between adjacent points on the same trajectory as discussed above. In this section, we examine the selection of spatio-temporal neighbors between trajectories.

In keeping with the framework established by Isomap, all neighbor links are bidirectional. The typical Euclidean metric is used to calculate the distance between neighboring points, and the geodesic distance between all other pairs is the shortest distance along links in the graph.

When searching for a point's neighbors, individual points are not considered in isolation. Instead, we consider each pair of trajectories separately, and search for subsequences of those trajectories that contain points that match in a roughly pairwise manner. That is, the indices in both subsequences increase in the same direction, and both subsequences contain approximately the same number of points. Since all trajectories are initially subsampled at a uniform distance between adjacent points, this mean that, informally, matching subsequences travel in the same direction.

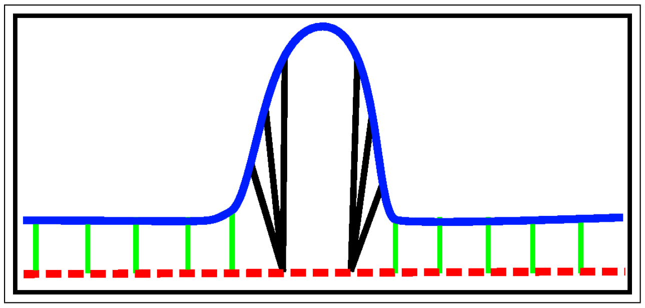


Fig. 4. Many-to-one neighbor links result in a graph with too many links, and thus geodesic distances that are too small. Deviations such as that shown in the solid (blue) trajectory should appear distant in the neighbor graph.

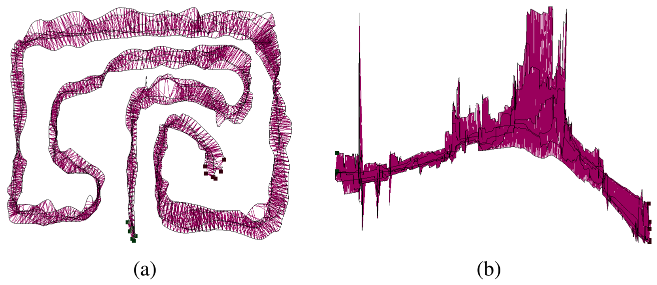


Fig. 5. A two-dimensional projection of the workspace trajectories for the wire maze task (left) and the two-dimensional embedding of its 7-DOF configuration space (right). Purple lines represent neighbor links.

Noisy links are detected and removed by searching for sequences of neighbor links in one trajectory whose indices in the matching trajectory do not monotonically increase. This ensures that adjacent subsequences in one trajectory are not connected to distant portions of the other trajectory, as may occur when a trajectory contains a loop (see figure 2) or other artifact (see figure 3). Finally, one-to-many neighbor links are not allowed. Only the one-to-one link with the shortest distance is permitted in the final neighbor graph. This restriction ensures that the geodesic distances increase quickly when trajectories deviate from one another (see figure 4). This strategy is illustrated further in the experimental results in section IV.

Additional checks may be incorporated into this strategy to restrict the types of neighbor links formed. For example, given an explicit or implicit environment model, as discussed above, we might check that a motion between every pair of neighbors is safe to execute. If the robot would impact an obstacle, that link may be removed from the graph. This helps ensure that, in the embedding, movement between nearby points is safe.

C. Planning

A robot is clearly capable of performing a demonstrated task by simply repeating example trajectories. However, this is undesirable for several reasons. Even expert robot operators are unlikely to produce perfect examples. Accidental movements, unavoidable jitter, detours, and sensor errors can all contribute to variations between demonstrations. In addition, the robot should have a safe strategy for operation if its position deviates from demonstrations due to sensing or actuation noise, or even due to the variety of initial conditions.

Planning requires a generalization of the provided example trajectories. Extrapolating cannot be safe without additional information about obstacles in the environment, but interpolation is possible if we know which portions of the demonstrated trajectories occur at the same point in the task. We argued in the previous section for the use of global information about trajectories to make this determination. Figure 5(a) shows a neighbor graph for the maze of figure 1. The graph alone is not sufficient to determine an action policy for undemonstrated points in the configuration space, though. Instead, we use the neighborhood information in the graph to create a simple low-dimensional space that facilitates interpolation of demonstrated actions.

MDS, the final step of Isomap, is used to create a two dimensional embedding such as the one shown in figure 5(b). This embedding preserves (as much as possible) the geodesic distances between all pairs of points. Although figure 5(a) illustrates a two-dimensional projection of the Cartesian workspace trajectories of the end effector, planning must occur in the seven-dimensional configuration space. Because the manipulator is redundant, it is necessary to be able to distinguish between different configurations that result in the same end-effector pose. In the embedding of figure 5(b), the starting points of the trajectories (the dark lines) are clustered along the left edge of the image. The trajectories end near the bottom-right of the image. The thinner, purple lines are neighbor links between trajectory points. Although nothing in the algorithm explicitly forces it to be so, the horizontal axis is roughly equivalent to time. This is because MDS selects the dimension with the greatest variance as the first dimension in the embedding. The vertical axis separates trajectories from one another. Vertical spikes represent portions of example trajectories that deviated from neighbor trajectories, similar to the examples in figures 2 and 4. Since the problematic spatio-temporal links pictured there were eliminated, the geodesic distances between trajectories increases significantly, and the trajectories become distant in the embedding.

Our previous work[24] investigated a strategy for planning in embedded spaces such as these. However, with the improved neighbor-selection mechanism presented here, the embedding creates a space in which planning is straightforward. A roughly linear path through this space from left to right, remaining in the area between example trajectories, is one simple strategy for generating novel plans. Future work will focus on other strategies, and their relative merits. For example, planning a path which remains near the densest areas of example trajectories may produce a plan more qualitatively similar to that desired by the user.

Trajectories created in this two-dimensional planning space must be transformed to the configuration space of the robot before they can be executed, but there is no global linear transformation between these spaces. Instead,

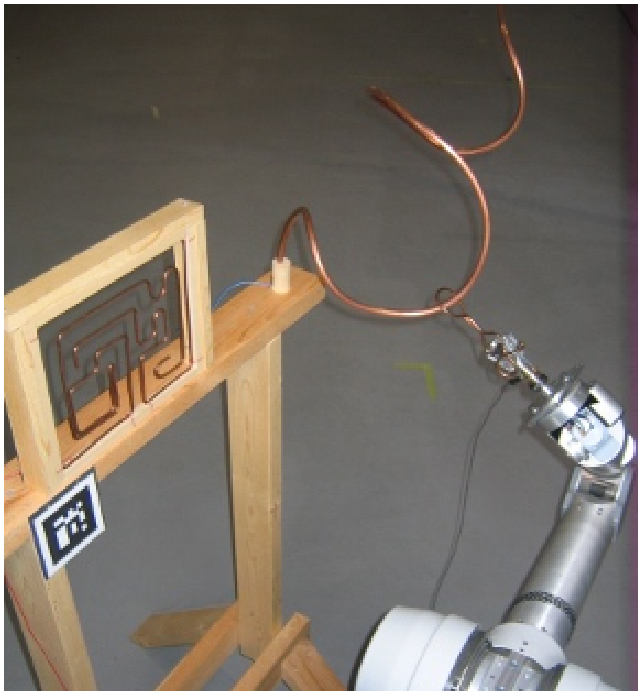


Fig. 6. The WAM manipulator, the wire maze (left) and the tube maze (right). The 2-D barcode in the lower-left is a visual fiducial used to detect the location of the rig relative to the robot. The rig and mazes are not modelled by the learning algorithm.

individual points in the planned path must be *lifted* to the original high-dimensional space. This is accomplished using the Delaunay triangulation [25] of the trajectory points embedded in two-dimensions. For any query point in the plane, the unique enclosing triangle is found. The barycentric coordinates of the query point within the triangle are used as weights to interpolate between the points corresponding to the triangle vertices in the original space. It should be noted that the mapping from the planning space to the original space is naturally a one-to-many mapping. However, since the correspondence between points in the planning space and the original points in the higher dimensional space is known, interpolation ensures that the resulting point is in the correct region of the configuration space.

This method is reliable only for query points that lie within the triangulation of the points of the example trajectories. Fortunately, this is precisely the area in which we can be confident executing novel plans. Points outside the triangulation cannot be lifted by interpolating between example points, and thus represent extrapolations outside the demonstrated area of the configuration space. A similar method may be used to map points in the other direction, from the configuration space to the planning space. This mapping is required to query the plan for an action to perform at a given configuration.

IV. Experimental Evaluation

Experiments were conducted using the 7-DOF WAM manipulator shown in figure 6 on two similar tasks. When the arm is operated in gravity-compensation mode, it can

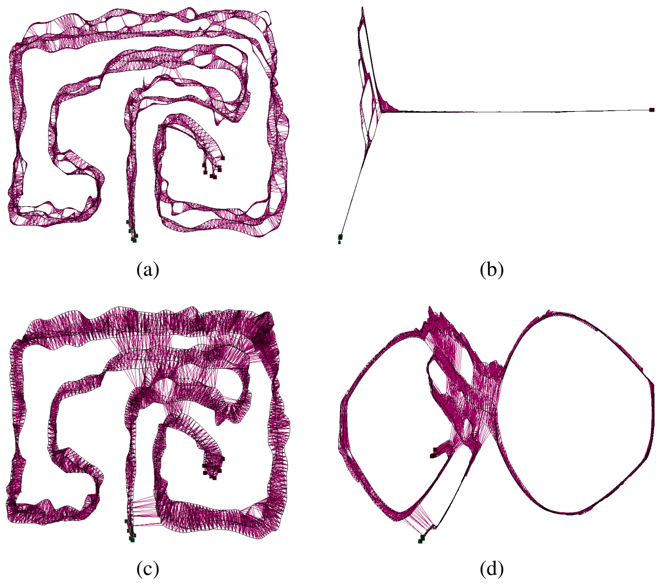


Fig. 7. Neighbors selected by (a) k-NN and (c) ST-Isomap, and their embeddings (b) and (d). Spurious short-circuit neighbor links produce embeddings unusable for planning.

be easily moved by hand. Participants were asked to perform kinesthetic demonstrations navigating the two mazes pictured. When the robot's copper end effector contacts the walls of either maze, a buzzer provides auditory feedback. The wire maze on the left is effectively two-dimensional, though some of the rotational axes are relatively unconstrained, allowing additional variation in the demonstrated trajectories. This rotational variation is not strictly necessary for navigating the maze and is unlikely to be correlated between example trajectories. The linear portion of the end effector is used to navigate the maze, and the proximal and distal loops keep the end effector within the plane of the maze. The tube maze on the right more fully explores the six degrees of freedom of the workspace. In this task, the loop at the tip of the end effector must be threaded over the copper tube to reach its base. Six demonstrations were performed on each maze by each participant.

Figure 7 shows the neighbor links chosen by other methods and the resulting embeddings. Figures (a) and (b) were produced using the k-nearest neighbor strategy of the original Isomap algorithm, with $k = 10$ chosen to ensure the number of neighbors per point is roughly equivalent to that of the other algorithms. The workspace plot of figure 7(a) appears sparser than the corresponding images for ST-Isomap (figure 7(c)) and our algorithm (figure 5(a)) because k-NN produces more temporal neighbors than spatio-temporal neighbors. The ST-Isomap results illustrate some of the most difficult situations for neighbor selection. In many cases, such as the spurious neighbor links near the bottom of figure 7(c), links are formed between trajectories that are relatively near to one another, and even travelling in parallel directions. Without considering global information about the trajectories, these incorrect links are difficult to detect.

The trajectories learned, even by our simple initial approach, appear qualitatively sound, and have been successfully executed on both mazes. In addition, the planned trajectories avoided joint limits more effectively than the example trajectories. During demonstration, users often rotated arm joints to their extremes even when this was not necessary to complete the task. Because the planning method relies on interpolation of demonstrations, planned trajectories remain away from joint limits whenever allowed by the demonstration data.

V. Conclusions and Future Work

In this paper, we presented a method for embedding robot trajectories in a low-dimensional space and a simple technique for creating novel plans in this space. Our work extends Isomap by introducing a neighbor-finding technique suited for time-series data such as configuration-space trajectories, and is free from the parameter selection required for the application of other approaches to multiple domains. This approach allows dimensionality reduction to produce a two-dimensional task-specific space in which safe planning is straightforward, even without a model of the environment in which the robot operates.

Future work will focus on development and evaluation of planning techniques in the embedded space, and possibly over the neighborhood graph itself. We also plan to address computational inefficiencies in the current approach. Since the embedding is not a globally linear transformation, straight lines are not preserved, and a large number of points must be lifted to faithfully transfer a plan from the embedded space to the original configuration space. By identifying regions of the demonstrations where less detail is required, such as areas of low diversity or low curvature, planning may be simplified in these areas.

Finally, we expect the improved neighbor-finding approach presented here to be useful in other applications. In addition to programming by demonstration, this technique may prove useful for activity recognition, robot fault detection, and other applications in which time-series trajectories are compared.

References

[1] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, May 1994, pp. 3310 – 3317.

[2] S. M. Lavalle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[3] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *Proceedings of the International Conference on Robotics and Automation*, San Diego, CA, 1994, pp. 2138–2145.

[4] M. Stolle and C. Atkeson, "Policies based on trajectory libraries," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 2006, pp. 3344–3349.

[5] R. Glaubius, M. Namihira, and W. D. Smart, "Speeding up reinforcement learning using manifold representations: Preliminary results," in *Proceedings of the IJCAI 2005 Workshop on Reasoning with Uncertainty in Robotics (RUR 05)*, Edinburgh, Scotland, July 2005.

[6] D. Bentivegna and C. Atkeson, "Learning from observation using primitives," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, 2001, pp. 1988–1993 vol.2.

[7] A. Ijspeert, J. Nakanishi, and S. Schaal, "Trajectory formation for imitation with nonlinear dynamical systems," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2, 2001, pp. 752–757 vol.2.

[8] C. Lee, "A phase space spline smoother for fitting trajectories," *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, vol. 34, pp. 346–356, 2004.

[9] A. Ude, C. Atkeson, and M. Riley, "Planning of joint trajectories for humanoid robots using b-spline wavelets," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 3, 2000, pp. 2223–2228 vol.3.

[10] J. Aleotti, S. Caselli, and G. Maccherozzi, "Trajectory reconstruction with nurbs curves for robot programming by demonstration," in *Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings. 2005 IEEE International Symposium on*, 2005, pp. 73–78.

[11] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, May 2007.

[12] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical system modulation for robot learning via kinesthetic demonstrations," *IEEE Transactions on Robotics*, 2008.

[13] N. Ratliff, D. Bradley, J. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press, 2007.

[14] H. Friedrich, J. Holle, and R. Dillmann, "Interactive generation of flexible robot programs," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, 1998, pp. 538–543 vol.1.

[15] B. Argall, B. Browning, and M. Veloso, "Learning by demonstration with critique from a human teacher," in *ACM/IEEE international conference on Human-robot interaction*. Arlington, Virginia, USA: ACM Press, 2007, pp. 57–64.

[16] N. Delson and H. West, "Robot programming by human demonstration: adaptation and inconsistency in constrained motion," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1, 1996, pp. 30–36 vol.1.

[17] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, Dec. 2000.

[18] K. Dautenhahn and C. L. Nehaniv, *Imitation in Animals and Artifacts*. MIT Press, 2002.

[19] O. C. Jenkins and M. J. Matarić, "A spatio-temporal extension to isomap nonlinear dimension reduction," in *The Twenty-first International Conference on Machine Learning*. Banff, Alberta, Canada: ACM Press, 2004, p. 56.

[20] A. Tsoli and O. C. Jenkins, "2d subspaces for user-driven robot grasping," in *Robotics, Science and Systems Conference: Workshop on Robot Manipulation*, 2007.

[21] M. Hein and M. Maier, "Manifold denoising as preprocessing for finding natural representations of data," in *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*. Menlo Park, CA: AAAI Press, Jul. 2007, pp. 1646–1649.

[22] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, B. Schlkopf, and B. S. Olkopf, "Learning with local and global consistency," *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 16*, vol. 16, pp. 321—328, 2003.

[23] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," in *Exploring artificial intelligence in the new millennium*. Morgan Kaufmann Publishers Inc., 2003, pp. 239–269.

[24] N. A. Melchior and R. Simmons, "Learning sequential composition plans using reduced-dimensionality examples," in *Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-01*. American Association for Artificial Intelligence, 2009.

[25] M. Bern and D. Eppstein, "Mesh generation and optimal triangulation," *Computing in Euclidean Geometry*, vol. 1, pp. 23–90, 1992.

REPORT OF INVENTIONS AND SUBCONTRACTS

(Pursuant to "Patent Rights" Contract Clause) (See Instructions on back)

Form Approved OMB No. 9000-0095 Expires Oct 31, 2004

PLEASE DO NOT RETURN YOUR COMPLETED FORM TO THIS ADDRESS. RETURN COMPLETED FORM TO THE CONTRACTING OFFICER.

1.a. NAME OF CONTRACTOR/SUBCONTRACTOR: Carnegie Mellon University	c. CONTRACT NUMBER: FA2386-10-1-4138	2.a. NAME OF GOVERNMENT PRIME CONTRACTOR: same	c. CONTRACT NUMBER:
b. ADDRESS *(Include ZIP Code)*: 5000 Forbes Avenue Pittsburgh, PA 15213-3815	d. AWARD DATE *(YYYYMMDD)*: 20100824	b. ADDRESS *(Include ZIP Code)*: same	d. AWARD DATE *(YYYYMMDD)*:

3. TYPE OF REPORT *(X one)*: a. INTERIM [✓ b. FINAL]

4. REPORTING PERIOD *(YYYYMMDD)*: a. FROM 20100824 b. TO 20130823

SECTION I - SUBJECT INVENTIONS

5. "SUBJECT INVENTIONS" REQUIRED TO BE REPORTED BY CONTRACTOR/SUBCONTRACTOR *(If "None," so state)*

NAME(S) OF INVENTOR(S) *(Last, First, Middle Initial)* a.	TITLE OF INVENTION(S) b.	DISCLOSURE NUMBER, PATENT APPLICATION SERIAL NUMBER OR PATENT NUMBER c.	ELECTION TO FILE PATENT APPLICATIONS (X) d. (1) UNITED STATES (a) YES	(1) UNITED STATES (b) NO	(2) FOREIGN (a) YES	(2) FOREIGN (b) NO	CONFIRMATORY INSTRUMENT OR ASSIGNMENT FORWARDED TO CONTRACTING OFFICER (X) e. (a) YES	e. (b) NO
None	None							

f. EMPLOYER OF INVENTOR(S) NOT EMPLOYED BY CONTRACTOR/SUBCONTRACTOR

(1) (a) NAME OF INVENTOR *(Last, First, Middle Initial)*	(2) (a) NAME OF INVENTOR *(Last, First, Middle Initial)*
(b) NAME OF EMPLOYER	(b) NAME OF EMPLOYER
(c) ADDRESS OF EMPLOYER *(Include ZIP Code)*	(c) ADDRESS OF EMPLOYER *(Include ZIP Code)*

g. ELECTED FOREIGN COUNTRIES IN WHICH A PATENT APPLICATION WILL BE FILED

(1) TITLE OF INVENTION	(2) FOREIGN COUNTRIES OF PATENT APPLICATION

SECTION II - SUBCONTRACTS *(Containing a "Patent Rights" clause)*

6. SUBCONTRACTS AWARDED BY CONTRACTOR/SUBCONTRACTOR *(If "None," so state)*

NAME OF SUBCONTRACTOR(S) a.	ADDRESS *(Include ZIP Code)* b.	SUBCONTRACT NUMBER(S) c.	FAR "PATENT RIGHTS" d. (1) CLAUSE NUMBER	d. (2) DATE *(YYYYMM)*	DESCRIPTION OF WORK TO BE PERFORMED UNDER SUBCONTRACT(S) e.	SUBCONTRACT DATES *(YYYYMMDD)* f. (1) AWARD	f. (2) ESTIMATED COMPLETION

SECTION III - CERTIFICATION

7. CERTIFICATION OF REPORT BY CONTRACTOR/SUBCONTRACTOR *(Not required if: (X as appropriate))* SMALL BUSINESS or NONPROFIT ORGANIZATION

I certify that the reporting party has procedures for prompt identification and timely disclosure of "Subject Inventions," that such procedures have been followed and that all "Subject Inventions" have been reported.

a. NAME OF AUTHORIZED CONTRACTOR/SUBCONTRACTOR OFFICIAL *(Last, First, Middle Initial)*	b. TITLE	c. SIGNATURE	d. DATE SIGNED
Simmons, Reid	Professor		January 3, 2014

DD FORM 882, DEC 2001 PREVIOUS EDITION IS OBSOLETE.

Reset

DD FORM 882 INSTRUCTIONS

GENERAL

This form is for use in submitting INTERIM and FINAL invention reports to the Contracting Officer and for use in reporting the award of subcontracts containing a "Patent Rights" clause. If the form does not afford sufficient space, multiple forms may be used or plain sheets of paper with proper identification of information by item number may be attached.

An INTERIM report is due at least every 12 months from the date of contract award and shall include (a) a listing of "Subject Inventions" during the reporting period, (b) a certification of compliance with required invention identification and disclosure procedures together with a certification of reporting of all "Subject Inventions," and (c) any required information not previously reported on subcontracts containing a "Patent Rights" clause.

A FINAL report is due within 6 months if contractor is a small business firm or domestic nonprofit organization and within 3 months for all others after completion of the contract work and shall include (a) a listing of all "Subject Inventions" required by the contract to be reported, and (b) any required information not previously reported on subcontracts awarded during the course of or under the contract and containing a "Patent Rights" clause.

While the form may be used for simultaneously reporting inventions and subcontracts, it may also be used for reporting, promptly after award, subcontracts containing a "Patent Rights" clause.

Dates shall be entered where indicated in certain items on this form and shall be entered in six or eight digit numbers in the order of year and month (YYYYMM) or year, month and day (YYYYMMDD). Example: April 1999 should be entered as 199904 and April 15, 1999 should be entered as 19990415.

1.a. Self-explanatory.

1.b. Self-explanatory.

1.c. If "same" as Item 2.c., so state.

1.d. Self-explanatory.

2.a. If "same" as Item 1.a., so state.

2.b. Self-explanatory.

2.c. Procurement Instrument Identification (PII) number of contract (DFARS 204.7003).

2.d. through 5.e. Self-explanatory.

5.f. The name and address of the employer of each inventor not employed by the contractor or subcontractor is needed because the Government's rights in a reported invention may not be determined solely by the terms of the "Patent Rights" clause in the contract.

Example 1: If an invention is made by a Government employee assigned to work with a contractor, the Government rights in such an invention will be determined under Executive Order 10096.

Example 2: If an invention is made under a contract by joint inventors and one of the inventors is a Government employee, the Government's rights in such an inventor's interest in the invention will also be determined under Executive Order 10096, except where the contractor is a small business or nonprofit organization, in which case the provisions of 35 U.S.C. 202(e) will apply.

5.g.(1) Self-explanatory.

5.g.(2) Self-explanatory with the exception that the contractor or subcontractor shall indicate, if known at the time of this report, whether applications will be filed under either the Patent Cooperation Treaty (PCT) or the European Patent Convention (EPC). If such is known, the letters PCT or EPC shall be entered after each listed country.

6.a. Self-explanatory.

6.b. Self-explanatory.

6.c. Self-explanatory.

6.d. Patent Rights Clauses are located in FAR 52.227.

6.e. Self-explanatory.

6.f. Self-explanatory.

7. Certification not required by small business firms and domestic nonprofit organizations.

7.a. through 7.d. Self-explanatory.